Generación de Código Objeto

El Generador de Código Objeto (en adelante **GCO**) va recibiendo las instrucciones del programa intermedio (en nuestro caso, **cuartetos**) y las va traduciendo a código objeto (en nuestro caso usaremos el lenguaje **ensamblador** ENS2001 como objeto, que después se ensamblará para obtener el código máquina ejecutable).

Se diseñará el GCO más sencillo, que es el que traduce usando una **plantilla** o *template* para cada instrucción intermedia: a cada código de operación le corresponde una traducción fija. Por ejemplo, de una manera simplificada, la traducción de la suma sería la mostrada en la Figura 1.

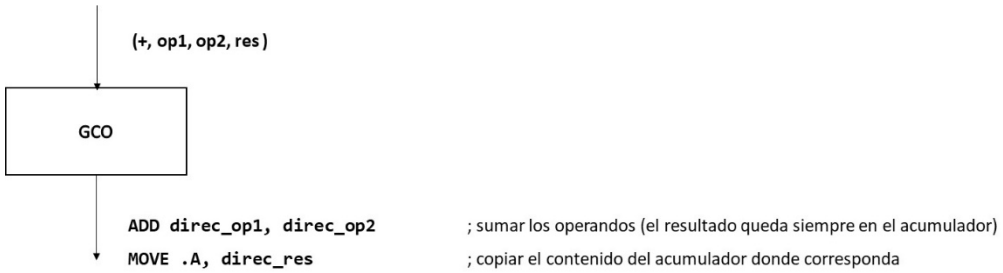

Figura 1: Ejemplo de plantilla de traducción para la instrucción suma de 3 direcciones

Es importante notar que los operandos de una instrucción pueden ser de distinta naturaleza (variable local, variable global, número entero, parámetro por referencia, etc.). Por tanto, la dirección del operando será diferente en cada uno de esos casos y el acceso a dicha dirección requerirá de un código diferente en cada uno de esos casos.

Dirección de las variables

Para cada variable que aparezca en el programa intermedio, en el programa objeto se tendrá la **dirección de la variable**, la cual depende de su ámbito (también llamado alcance o visibilidad). Se consideran 3 tipos de alcance:

- **Global**: Las variables globales se encuentran en memoria estática. Su dirección es relativa al inicio de la zona de variables estáticas.
- **Local**: Las variables locales, parámetros y variables temporales se encuentran en el registro de activación (en adelante RA) de la función que se está ejecutando. En la estrategia de asignación mediante pila, se encuentra en el RA que está en la cima de la pila, y su dirección es relativa al puntero de pila.
- **No locales**: Las variables no locales se encuentran en un registro de activación diferente del de la función que se está ejecutando. En la estrategia de asignación mediante pila, la variable está en un RA de la pila pero no en el de la cima, y se accede a ella mediante direccionamiento indirecto sobre el puntero de acceso (en adelante, PA). Este tipo de variables se tienen en lenguajes que permiten anidamiento de funciones, es decir, que permiten declarar una función dentro de otra función.

Ámbito global

La dirección de una variable global es **relativa al inicio de la zona de datos estáticos**. En dicha zona estarán todas las variables globales, una detrás de otra tal cual están en la Tabla de Símbolos (en adelante TS), así que la dirección de cada una de ellas es la dirección de inicio de la zona de datos estáticos más el desplazamiento de la variable en la TS. Supóngase una variable *x* que tiene un desplazamiento de 20 en la TS (Figura 2). Su dirección será "inicio_estáticas + 20".

En ENS2001, se podría usar el registro índice `IY` para marcar la zona de inicio de datos estáticos, de forma que la dirección de las variables globales sea relativa al registro `IY`. Esto se podría hacer definiendo una etiqueta que apunte a la primera posición de la zona de datos estáticos (Figura 3) y guardándola en `IY` (se supone que el conjunto de las variables globales ocupa 200 palabras):

```
MOVE #inicio_estaticas, .IY     ; al principio del programa
```

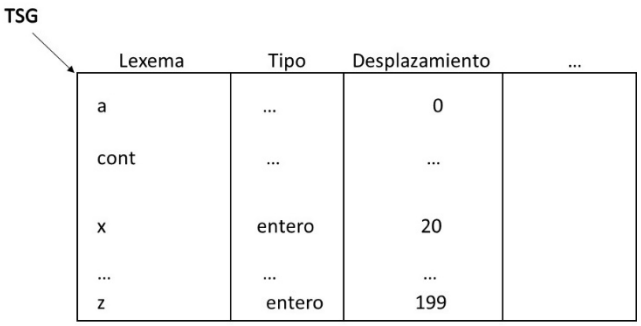

Figura 2: Tabla de Símbolos Global (tiempo de compilación)

Para que esta etiqueta tenga el valor adecuado (la dirección de la primera posición libre justo detrás el código máquina), se puede usar la siguiente pseudoinstrucción, ubicada al final del código objeto:

```
inicio_estaticas: RES 200 ; crea la etiqueta apuntando a la
                          ; primera de las 200 posiciones que reserva
```

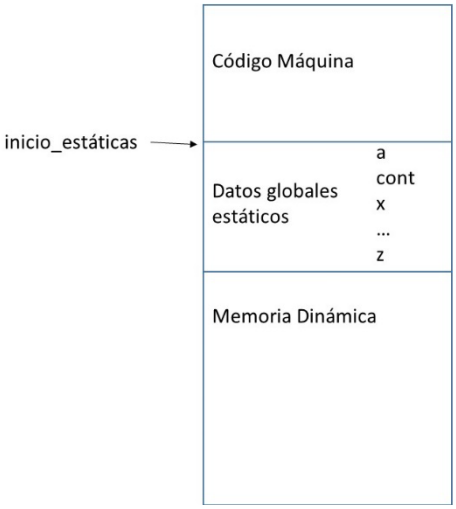

Figura 3: Organización de la memoria para la ejecución

Con todo lo explicado, **la dirección de la variable *x*** sería **#20[.IY]**

Así, por ejemplo, la traducción de la instrucción de código intermedio **x:= 0**, cuando *x* es la variable global anteriormente descrita, sería[1]:

```
MOVE #0, #20[.IY]
```

Ámbito local

Variable local

La dirección de una variable local, considerando la estrategia de asignación de memoria a los RA mediante pila, es **relativa al puntero de pila**. En el RA estarán todas las variables locales, una detrás de otra, tal cual se indica en la Tabla de Símbolos (TS). Supóngase una variable *x* que tiene un desplazamiento de 20 en la TS (Figura 4). Su dirección será un desplazamiento relativo de "20 + t" respecto al puntero de cima de pila, siendo t el espacio que hay entre el puntero de pila y el inicio de la zona de parámetros y variables locales en el RA (donde normalmente se guardarían el Estado de la Máquina (EM), el Puntero de Acceso (PA) y el Puntero de Control (PC), si fueran necesarios).

[1] Se ha elegido esta instrucción por su sencillez, aunque no es una instrucción que vaya a aparecer en el código intermedio tal como se ha realizado la generación del código intermedio en las Traducciones Dirigidas por la Sintaxis.

En ENS2001, se podría usar el registro índice IX como puntero de pila (ya que solo IX e IY permiten hacer direccionamiento relativo).

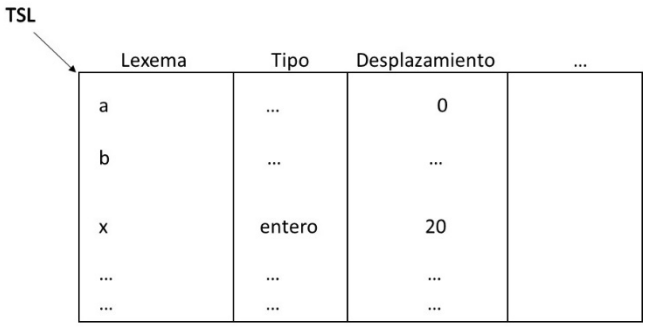

Figura 4: Una Tabla de Símbolos local (tiempo de compilación)

Hay que tener en cuenta que, antes de la primera llamada a función, habrá que haber determinado dónde comienza la pila, para lo cual podemos usar una etiqueta y asignársela al puntero de cima de pila:

```
MOVE #inicio_pila, .IX    ; se usará IX como puntero de pila
```

Para que esta etiqueta tenga el valor adecuado (la dirección de la primera posición de memoria dinámica, justo a continuación de los datos globales estáticos), se puede usar la siguiente pseudoinstrucción, ubicada al final del código objeto justo detrás de la instrucción que reserva el espacio para los datos estáticos):

```
inicio_pila: NOP    ; crea la etiqueta y le asigna la dirección en la que está
                    ; ubicada esta instrucción en el código ensamblador
```

Por tanto, **la dirección de la variable *x*** sería **#23[.IX]** (si el RA contiene PC) o **#22[.IX]** (si no se tiene PC). El diseño del RA es conocido en tiempo de compilación (Figura 5), así que *t* es conocido y se suma en compilación (para generar la instrucción ensamblador) al valor de desplazamiento que se obtiene al consultar la TS, es decir, se suma a lo que devuelve la función *BuscaDesplTS(x)*.

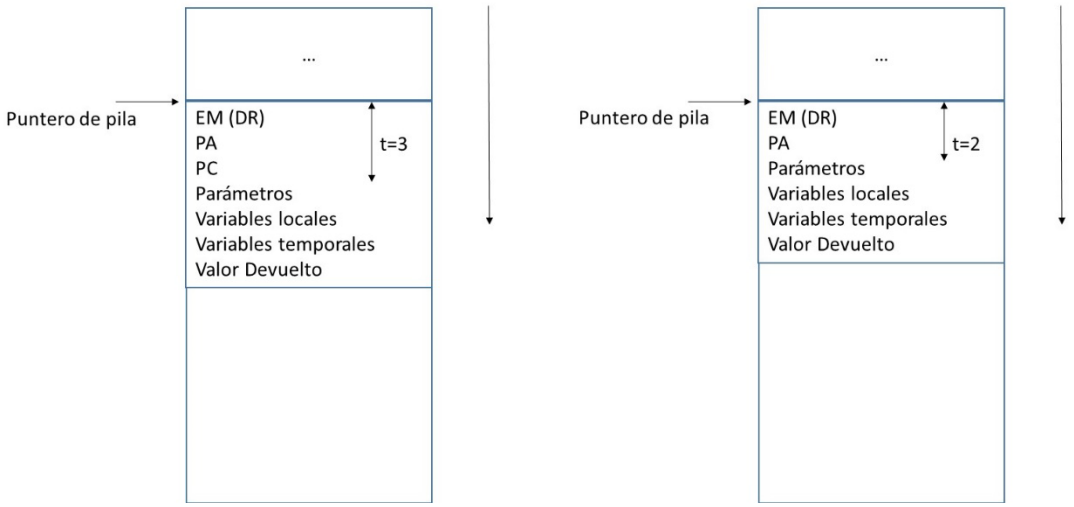

Figura 5: Diseño del Registro de Activación (en pila) con o sin Puntero de Control

Así, por ejemplo, la traducción de la instrucción **x:= 0**, cuando *x* es la variable local anteriormente descrita, sería[2]:

```
MOVE #0, #22[.IX]
```

[2] Se ha elegido esta instrucción por su sencillez, aunque no es una instrucción que vaya a aparecer en el código intermedio tal como se ha realizado la generación del código intermedio en las Traducciones Dirigidas por la Sintaxis.

Variable temporal

En la solución propuesta, las variables temporales reciben el mismo tratamiento que las variables locales: están en el RA activo (y en tiempo de compilación, en la Tabla de Símbolos) y su dirección es relativa al puntero de cima de pila.

Parámetro por valor

Un parámetro por valor recibe el mismo tratamiento que una variable local: está almacenado en el RA activo y, por tanto, su dirección es relativa al puntero de cima de pila.

Por ejemplo, suponiendo un parámetro formal *x* que se pasa por valor, que tiene un desplazamiento 10, que el Estado de la Máquina y el Puntero de Acceso ocupan en total 2 palabras, que no hay Puntero de Control y que IX se está usando como Puntero de Pila, **la dirección de *x*** sería **`#12[.IX]`**.

Parámetro por referencia

Un parámetro que se pasa por referencia requiere de un tratamiento específico ya que lo que se tiene en el RA es la dirección en la cual está realmente almacenado. Por tanto, se necesita realizar una indirección para poder acceder su valor.

Por ejemplo, suponiendo un parámetro formal *x* que se pasa por referencia, que tiene un desplazamiento 10, que el Estado de la Máquina y el Puntero de Acceso ocupan en total 2 palabras, que no hay Puntero de Control y que IX se está usando como Puntero de Pila, **la dirección de *x*** se podría obtener mediante:

```
MOVE #12[.IX], .R9  ; se accederá a x mediante la indirección sobre R9
```

Ahora que R9 contiene la dirección donde está el valor de *x*, **la dirección de la variable** sería **`[.R9]`**.

Así, por ejemplo, la traducción de la instrucción **x:= 0**, cuando *x* es el parámetro por referencia anteriormente descrito, sería[3]:

```
MOVE #12[.IX], .R9
MOVE #0, [.R9]
```

Ámbito no local

Para acceder a una variable no local, considerando la estrategia de asignación de memoria a los RA mediante pila, se realiza un direccionamiento indirecto sobre el PA de la forma que se explica continuación.

¿Qué ha de ocurrir en ejecución? Si se supone que la función que se está ejecutando, y cuyo RA está en la cima de la pila, usa una variable *x* que se ha declarado en otra función que está 3 niveles de profundidad por encima de la función que se está ejecutando (de nuevo se supone que el desplazamiento de *x* en su TS es 20 (Figura 4) y esta vez se prescindirá ya del Puntero de Control), la situación sería la que se refleja en la Figura 6 (el acceso a *x* requiere de 3 indirecciones sobre el PA más un desplazamiento):

La **dirección de la variable *x* no local** se puede escribir de la siguiente manera, usando ENS2001:

```
MOVE .IX, .R9          ; se salva el puntero de pila para no perderlo
MOVE #1[.IX], .IX      ; en IX estará el PA del padre (1980)
MOVE [.IX], .IX        ; en IX estará el PA del abuelo (1900)
MOVE [.IX], .IX        ; en IX estará el PA del bisabuelo (1710)
```

Ahora que IX contiene el PA del RA en el que está *x*, **la dirección de la variable *x*** es **`#21[.IX]`**, donde 21 es la suma del desplazamiento de *x* en la TS y lo que ocupa el PA (el tamaño de *s* en la figura, que es el espacio que hay

[3] Se ha elegido esta instrucción por su sencillez, aunque no es una instrucción que vaya a aparecer en el código intermedio tal como se ha realizado la generación del código intermedio en las Traducciones Dirigidas por la Sintaxis.

entre el PA y el inicio de la zona de parámetros y variables locales en el RA). Una vez que se ha accedido a *x*, se restauraría IX para que vuelva a ser el puntero de pila.

```
MOVE .R9, .IX       ; restaura el puntero de pila
```

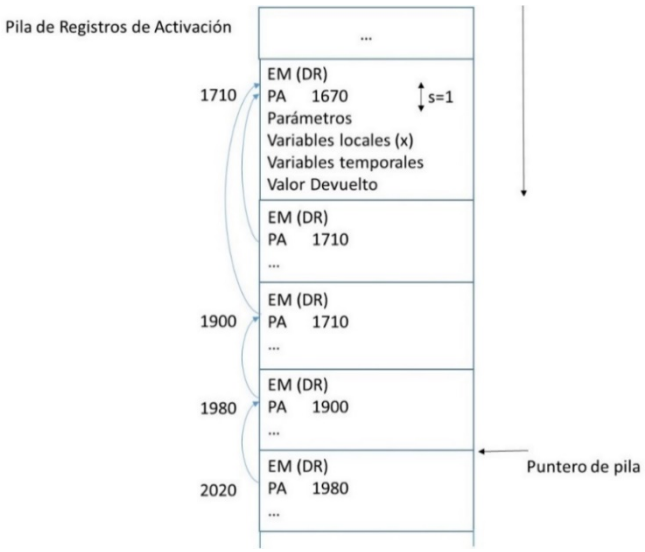

Figura 6: Pila de Registros de Activación

Así, por ejemplo, la traducción de la instrucción **x:= 0**, cuando *x* es la variable no local anteriormente descrita, sería[4]:

```
MOVE .IX, .R9
MOVE #1[.IX], .IX
MOVE [.IX], .IX
MOVE [.IX], .IX
MOVE #0, #21[.IX]
MOVE .R9, .IX
```

Traducción de las instrucciones del programa intermedio

Una vez sabido cómo se genera la dirección de cada variable, se tratará la traducción de las instrucciones. El generador de código objeto consulta el código de operación del cuarteto y realiza una traducción sistemática aplicando una plantilla o *template* para cada uno de los códigos de operación.

Se indican a continuación cómo podrían ser estas plantillas usando el ensamblador ENS2001, dejando de lado la cuestión de si los operandos y el resultado son variables (globales, locales o no locales) o son constantes numéricas o parámetros o cadenas de caracteres, etc. Cada uno de estos tipos de datos requieren tratamientos diferentes.

Hay que tener en cuenta que **no se va a usar el puntero de pila SP del ENS2001, ni se van a usar las instrucciones PUSH, POP, CALL y RET** (dado que modifican el puntero de pila), pues su funcionamiento no coincide con el modo de trabajo explicado aquí.

Instrucciones de operaciones aritméticas y lógicas

Cuarteto	Plantilla Traducción
(*, op1, op2, res)	MUL op1, op2 MOVE .A, res
(AND, op1, op2, res)	AND op1, op2 MOVE .A, res
…	…

[4] Se ha elegido esta instrucción por su sencillez, aunque no es una instrucción que vaya a aparecer en el código intermedio tal como se ha realizado la generación del código intermedio en las Traducciones Dirigidas por la Sintaxis.

Instrucción de asignación

Cuarteto	Plantilla Traducción
(:=, op1, , res)	`MOVE op1, res`

Instrucciones de bifurcación

Cuarteto	Plantilla Traducción
(goto, , , etiq)	`BR /etiq`
(if>, op1, op2, etiq)	`CMP op1, op2` `BP /etiq ; salto si positivo`
…	…

Instrucciones de paso de parámetros

Cuarteto	Plantilla Traducción
(param, , , op1)	`MOVE op1, #posición_parámetro[.IX]`
(param&, , , op1)	`MOVE /op1, #posición_parámetro[.IX]`

Hay que tener en cuenta que `#posición_parámetro` se obtendrá sumándole al `IX` el Tamaño del RA del que hace la llamada más el espacio que ocupan el Estado de la Máquina, el Puntero de Acceso y el Puntero de Control (si existen) más el desplazamiento del parámetro. Este desplazamiento del parámetro se tiene que calcular en función del número de parámetros que se hayan pasado ya al nuevo RA y de lo que ocupe cada uno de ellos.

Teniendo en cuenta que ENS2001 no permite realizar mediante una única instrucción todo lo indicado en el párrafo anterior, serían necesarias varias instrucciones para obtener la posición del parámetro. Por lo tanto, como no es posible hacer algo como "**`MOVE op1, #Tam_RA_llamador+2[.IX]`**", en su lugar podría hacerse:

```
ADD #Tam_RA_llamador, .IX
ADD #2, .A
MOVE op1, [.A]
```

Por otra parte, en el cuarteto *param&*, que corresponde al paso de parámetro por referencia, lo que hay que hacer es enviar la dirección del operando *op1* (que se ha representado en la plantilla como */op1* para diferenciarlo del caso anterior).

Instrucciones de llamada a función y retorno

Cuarteto	Plantilla Traducción
(call, et1_p, ,)	`; Llamada a la función p que no devuelve nada` `; almacena el EM (DR):` `MOVE #dir_ret1, #Tam_RA_llamador[.IX]` `; establecimiento del PA. Se genera código para recorrer` `; (prof`$_{\text{llamador}}$` – prof`$_{\text{llamado}}$` + 1) PA desde el llamador. Suponiendo que la` `; diferencia de profundidades sea 2, hay que recorrer 3 PA:` `MOVE #1[.IX], .R9 ; R9 apunta al PA del padre del llamador` `MOVE [.R9], .R9 ; se generarán prof`$_{\text{llamador}}$` – prof`$_{\text{llamado}}$` `; instrucciones como ésta.` `MOVE [.R9], .R9 ; R9 apunta al PA del abuelo, que es al que ha de` `; apuntar el PA del RA del llamado` `ADD #Tam_RA_llamador, .IX` `INC .A` `MOVE .R9, [.A] ; almacena el PA del llamado`  `; si hubiera Puntero de control, se almacenaría así:` `; INC .A` `; MOVE .A, .R9` `; ADD #2, .IX` `; MOVE .A, [.R9]`

	; se incrementa el puntero de pila: **ADD #Tam_RA_llamador, .IX** **MOVE .A, .IX** ; se salta al código de la función llamada: **BR /et1_p** ; se decrementa el puntero de pila: **dir_ret1: SUB .IX, #Tam_RA_llamador** **MOVE .A, .IX** ; si se tuviera Puntero de Control, se utilizaría para el retorno. ; En vez de decrementar el puntero de pila, se haría: ; **MOVE #2[.IX], .IX** ; IX apunta al PC del RA del llamador ; **SUB .IX, #2** ; **MOVE .A, .IX** ; IX apunta al RA del llamador
(call, et1_p, , t_i)	; Llamada a la función p que devuelve un valor que se almacena en t_i ; almacena el EM (DR): **MOVE #dir_ret1, #Tam_RA_llamador[.IX]** ; establecimiento del PA. Se genera código para recorrer ; ($prof_{llamador}$ – $prof_{llamado}$ + 1) PA desde el llamador. Suponiendo que la ; diferencia de profundidades sea 2, hay que recorrer 3 PA: **MOVE #1[.IX], .R9** ; R9 apunta al PA del padre del llamador **MOVE [.R9], .R9** ; se generarán $prof_{llamador}$ – $prof_{llamado}$; instrucciones como ésta. **MOVE [.R9], .R9** ; R9 apunta al PA del abuelo, que es al que ha de ; apuntar el PA del RA del llamado **ADD #Tam_RA_llamador, .IX** **INC .A** **MOVE .R9, [.A]** ; almacena el PA del llamado ; si hubiera Puntero de control, se almacenaría así: ; **INC .A** ; **MOVE .A, .R9** ; **ADD #2, .IX** ; **MOVE .A, [.R9]** ; se incrementa el puntero de pila: **ADD #Tam_RA_llamador, .IX** **MOVE .A, .IX** ; se salta al código de la función llamada: **BR /et1_p** ; el llamador recoge el valor devuelto por el llamado: **dir_ret1: SUB #Tam_RA_p, #1** ; Se resta el tamaño del valor devuelto (VD), que ; se conoce por la Tabla de Símbolos. Sería 1 para un entero, ; una dirección o un lógico, o > 1 para cadenas, reales… **ADD .A, .IX** ; el acumulador contiene la dirección del VD **MOVE [.A], .R9** ; R9 contiene el valor devuelto ; se decrementa el puntero de pila: **SUB .IX, #Tam_RA_llamador** **MOVE .A, .IX** ; se copia el valor devuelto en el temporal correspondiente **MOVE .R9, #32[.IX]** ; t_i:= VD (se ha supuesto que el ; desplazamiento de t_i en la TS es 32)

(return, , ,)	; Return de una función que no devuelve nada: **BR [.IX]** ; devuelve el control al llamador
(return, , ,)	; Return de la función principal: **HALT** ; detiene la ejecución del programa
(return, op1, ,)	; Return de una función p que devuelve op1: **SUB #Tam_RA_p, #1** ; Se resta el tamaño del valor devuelto, que se conoce por la Tabla de ; Símbolos. Sería 1 para un entero, una dirección o un lógico, o ; > 1 para cadenas, reales… **ADD .A, .IX** ; el acumulador contiene la dirección del valor devuelto **MOVE #48[.IX], [.A]** ; copia op1 en la dirección del campo valor devuelto. ; Se ha supuesto que el desplazamiento de op1 en la TS de p es 48 **BR [.IX]** ; devuelve el control al llamador

Nota sobre el establecimiento del Puntero de Acceso (PA)

El PA del Registro de Activación de una función apunta al campo PA de la activación más reciente de su bloque padre (concepto éste asociado al de anidamiento de funciones).

Como ejemplo, se va a suponer que la función *P* llama a la función *Q*. La función que hace la llamada (*P*) es responsable de establecer el Puntero de Acceso del Registro de Activación de la función llamada (*Q*). Por tanto, *P*:

1) identifica a qué dirección ha de apuntar el PA de *Q* (apuntará al campo PA de la activación más reciente del padre de *Q*). Para ello, la función *P* recorre, desde su propio PA, "$prof_P - prof_Q + 1$" punteros de acceso (este es el número de saltos o indirecciones a realizar). El código generado depende, por tanto, de la relación entre *P* y *Q*.
2) guarda esa dirección en el campo PA del RA de *Q*. El código generado es siempre el mismo, y no depende de la relación que exista entre *P* y *Q*. Suponiendo que está, por ejemplo, en R9:

```
ADD #Tam_RA_llamador, .IX
INC .A          ; sustituir por ADD #d,.A si el PA está a d posiciones del inicio del RA, en vez de a 1
MOVE .R9, [.A]
```

Los diferentes casos que se pueden dar para el paso 1 (y su correspondiente código en ENS2001) son los siguientes:

A. ***P* llama a su hija *Q***

Está claro que el PA de *Q* ha de apuntar al PA de *P*, que es su padre. Aplicando la teoría aprendida, la profundidad de *P* es *n*, y la de *Q* es *n+1*. La diferencia de profundidades más 1 es "$prof_P - prof_Q + 1 = n - (n+1) + 1 = 0$", por lo tanto, **no hay que realizar ningún salto**.

```
; identifica a qué dirección ha de apuntar el PA de Q:
ADD #1, .IX          ; el acumulador contiene la dirección del PA de P, que es a la que hay que apuntar
MOVE .A, .R9         ; salva en R9 la dirección a la que hay que apuntar
```

B. ***P* llama a su hermana *Q***

Está claro que el PA de *Q* ha de apuntar al PA del padre de *P* (ambos tienen el mismo padre). Aplicando la teoría aprendida, la profundidad de *P* es *n*, y la de *Q* es *n*. La diferencia de profundidades más 1 es "$prof_P - prof_Q + 1 = n - n + 1 = 1$", por lo tanto, **hay que realizar un salto** desde el PA de *P* para llegar al padre.

```
; identifica a qué dirección ha de apuntar el PA de Q:
MOVE #1[.IX], .R9    ; R9 contiene la dirección del PA del padre de P
```

C. ***P* es "nieta" o "bisnieta" o… de la función *Q* a la que llama**

Para el ejemplo se va a suponer que ***P* es nieta de *Q*** (entonces el padre de *Q* es el bisabuelo de *P*). Aplicando la teoría aprendida, la profundidad de *P* es *n*, y la de *Q* es *n–2*. La diferencia de profundidades más 1 es "$prof_P - prof_Q + 1 = n - (n-2) + 1 = 3$", y, por lo tanto, **hay que realizar tres saltos** desde el PA de *P* para llegar al padre de *Q*.

```
; identifica a qué dirección ha de apuntar el PA de Q:
MOVE #1[.IX], .R9    ; R9 contiene la dirección del PA del padre de P (primer salto).
                     ; A continuación, hay que generar el siguiente MOVE i veces,
                     ; siendo i la diferencia de profundidades
MOVE[.R9], .R9       ; R9 apunta ahora al abuelo
MOVE[.R9], .R9       ; R9 apunta ahora al bisabuelo
```

Cómo representar cada campo de los cuartetos

Teniendo en cuenta que el generador de código intermedio genera cuartetos, para poder implementar el generador de código objeto aquí explicado se va a repasar cómo representar en los cuartetos toda la información que necesita el generador de código objeto.

Un cuarteto tiene 4 campos: código de operación, operando 1, operando 2 y resultado.

El código de operación indica qué instrucción es, y **se implementará por sencillez mediante un código entero**. Como ya se ha visto, el generador de código objeto consulta el código de operación y aplica la correspondiente plantilla o *template* de traducción.

Para generar correctamente la dirección de los operandos y del resultado, hay que tener en cuenta de qué tipo son. Por lo tanto, **los operandos y el resultado serán estructuras, con dos campos: el tipo del operando y el operando** en sí. Los tipos habituales, cada uno de los cuales se implementarán por sencillez mediante un código entero, son los siguientes:

Tipos de los campos del cuarteto	**Observaciones**	**Código para el tipo**
Variable global	El operando en sí deberá tener información del desplazamiento	1
Variable local	El operando en sí deberá tener información del desplazamiento	2
Variable no local	El operando en sí deberá tener información del desplazamiento y de la diferencia de profundidades	3
Parámetro por valor	En nuestra solución, tiene el mismo tratamiento que una variable local	4 (= 2)
Parámetro por referencia	El operando en sí deberá tener información del desplazamiento	5
Variable temporal	En nuestra solución, tiene el mismo tratamiento que una variable local	6 (= 2)
Constante entera	El operando en sí deberá tener información del valor de la constante	7
Constante real	El operando en sí deberá tener información del valor de la constante	8
Constante cadena (*string*)	El operando en sí deberá contener la cadena	9
Constante carácter	El operando en sí deberá tener información del valor de la constante	10
Etiqueta	El operando en sí deberá contener la etiqueta	11
…		…