

# Tema 9:

## Generación de Código Intermedio

---

1. Introducción
2. Código de tres direcciones
3. Generación de código intermedio mediante gramáticas de atributos
4. Estructuras y elementos de una matriz
5. Expresiones lógicas
6. Referencias no satisfechas. Relleno por retroceso
7. Instrucciones de control de flujo
8. Llamadas a subprogramas

1

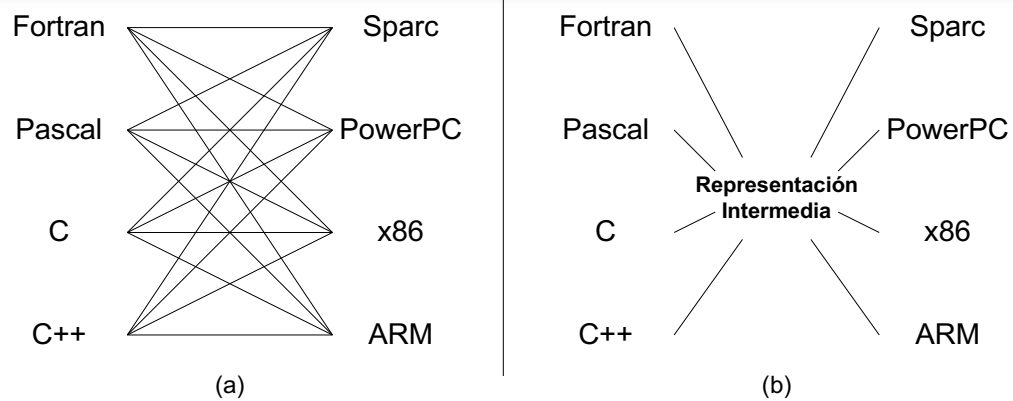
V. 16.2

---

## 1. Introducción

2

# Representaciones intermedias



## Tipos de representaciones intermedia

- Grafos Dirigidos Acíclicos (GDA)
- Árbol de sintaxis abstracta (AST)
- Formato SSA (Static Single-Assignment)
- Código de tres direcciones

3

## 2. Código de 3 direcciones

4

## Juego de instrucciones

---

Asignación	$x := y \text{ op } z$	
Salto incondicional	<b>goto</b> E	
Salto condicional	<b>if</b> x op y <b>goto</b> E	
Pila de activación	<b>push</b> x	
	x := <b>pop</b>	
Llamada y retorno de subprograma		<b>call</b> E <b>ret</b>
Asignaciones relativas	a [i] := x	
	x := a[i]	
Acceso a la pila	<b>FP</b>	
	<b>TOP</b>	

5

---

## 3. Generación de CI mediante gramáticas atribuidas

6

- Usaremos un procedimiento *emite* con un efecto colateral: Almacena código intermedio
- Ej. *emite* (x ':=' 5 '+' 9)
- *SIGINST*: Variable global con el número de la "SIGuiente INStrucción"

Asig  $\rightarrow$  id = E  
 E  $\rightarrow$  E + E  
 E  $\rightarrow$  id

Asig $\rightarrow$ id = E	{ Asig.pos := BuscaPos ( id.nom ) ; emite ( Asig.pos ':=' E.pos ); }
E $\rightarrow$ E <sub>1</sub> + E <sub>2</sub>	{ E.pos := CrearVarTemp() ; emite ( E.pos ':=' E <sub>1</sub> .pos '+' E <sub>2</sub> .pos ) }
E $\rightarrow$ id	{ E.pos := BuscaPos (id.nom ) }

7

## Árbol anotado

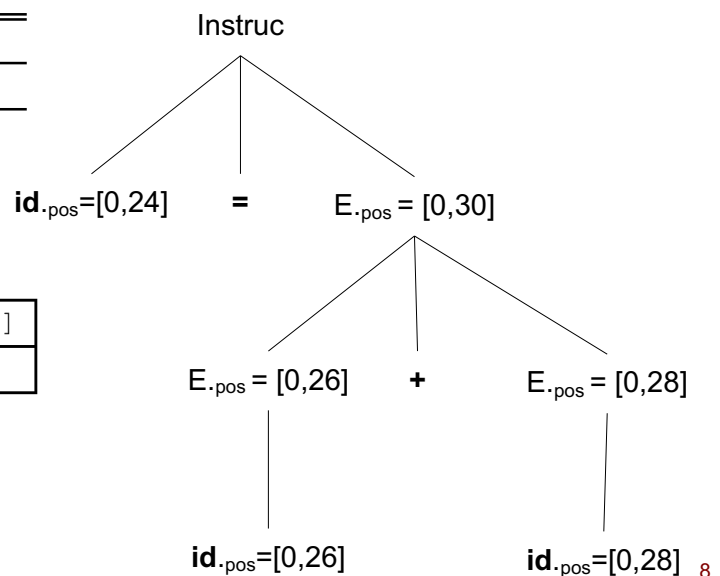
Cadena: a = b + c

Nom	tipo	posición [nivel, desp]
a	tentero	[0,24]
b	tentero	[0,26]
c	tentero	[0,28]

Código tres direcciones generado

(1) [0,30] := [0,26] + [0,28]
(2) [0,24] := [0,30]

(1) t1 := b + c
(2) a := t1



## 4. Estructuras y elementos de una matriz

9

### Acceso a miembros de una estructura

```
struct ej {  
    int c1 ;  
    float c2 ;  
}a, b;
```

TALLA_REAL = 4
TALLA_ENTERO = 2

La TDS quedaría:

TDS					Tabla de registros		
nom	tipo	posición	...	ref	nom	tipo	posición
a	testructura	0 , 124		●	c1	tinteger	0
b	testructura	0 , 130		●	c2	treal	2

10

## Acceso a miembros de una estructura

$E \rightarrow id_1.id_2$	<pre> { si BuscaTipo (id<sub>1</sub>.nom) &lt;&gt; testestructura <u>ent</u> MemError();   <u>sino</u> base := BuscaPos (id<sub>1</sub>.nom) ;     <u>si</u> no EsMiembro (id<sub>1</sub>.nom, id<sub>2</sub>.nom) <u>ent</u> MemError()     <u>sino</u> desp_miembro:=BuscaPosMiembro(id<sub>1</sub>.nom, id<sub>2</sub>.nom);       E.pos := base + desp_miembro ; } </pre>
$Asig \rightarrow id_1.id_2 = E$	<pre> { si BuscaTipo (id<sub>1</sub>.nom) &lt;&gt; testestructura <u>ent</u> MemError();   <u>sino</u> base := BuscaPos (id<sub>1</sub>.nom) ;     <u>si</u> no EsMiembro (id<sub>1</sub>.nom, id<sub>2</sub>.nom) <u>ent</u> MemError()     <u>sino</u> desp_miembro:=BuscaPosMiembro(id<sub>1</sub>.nom, id<sub>2</sub>.nom);       pos := base + desp_miembro ;       emite(pos ':=' E.pos); } </pre>

11

## Acceso a elemento de una matriz

Declaración:  $\text{int } A[n_1][n_2] \dots [n_n];$

Acceso al elemento  $A[i_1][i_2] \dots [i_n]$ :  $\text{base} + (((i_1 * n_2 + i_2) * n_3 + i_3 \dots) * n_n + i_n) * \text{Talla}$

$n_i$  es el número de elementos de la  $i$ -ésima dimensión

$\text{Inst\_simple} \rightarrow \mathbf{id}$	<pre>{ LI.nom := id.nom }</pre>
$\text{LI} = E$	<pre> { emite (LI.pos := LI.pos '*' Talla (id.nom)) ;   pos := BuscaPos (id.nom) ;   emite ( pos '[' LI.pos ']' := E.pos ; } </pre>
$\text{LI} \rightarrow [ E ]$	<pre> { LI.pos := CrearVarTemp() ;   emite (LI.pos := E.pos); LI.ndim := 1; } </pre>
$\text{LI} \rightarrow$	<pre>{ LI<sub>1</sub>.nom := LI.nom ; }</pre>
$\text{LI}_1 [ E ]$	<pre> { LI.ndim := LI<sub>1</sub>.ndim + 1; LI.pos := LI<sub>1</sub>.pos ;   emite (LI.pos := LI.pos '*' Num_elementos(LI.nom, LI.ndim)) ;   emite (LI.pos := LI.pos '+' E.pos) ; } </pre>

## 5. Expresiones lógicas

13

$E \rightarrow \mathbf{true}$	$\{$ E.pos:= CrearVarTemp() ; emite( E.pos ':=' 1 ) $\}$
$E \rightarrow \mathbf{false}$	$\{$ E.pos:= CrearVarTemp() ; emite( E.pos ':=' 0 ) $\}$
$E \rightarrow ( E_1 )$	$\{$ E.pos := E <sub>1</sub> .pos $\}$
$E \rightarrow \mathbf{id}$	$\{$ E.pos := BuscaPos(id.nom) $\}$

$E \rightarrow E_1 \mathbf{oprel} E_2$	$\{$ E.pos := CrearVarTemp() ; emite( E.pos ':=' 1 ) ; emite( 'if' E <sub>1</sub> .pos oprel.op E <sub>2</sub> .pos 'goto' SIGINST+ 2 ) ; emite( E.pos ':=' 0 ) $\}$
--	---

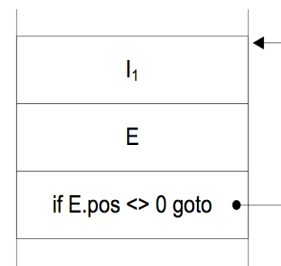
14

$E \rightarrow E_1 \text{ or } E_2$	<pre>{ E.pos:= CrearVarTemp() ;   emite( E.pos ':=' E1.pos '+' E2.pos )   emite( 'if' E.pos '&lt;= 1' goto' SIGINST+2 ) ;   emite( E.pos ':=' 1 ) }</pre>
$\rightarrow E_1 \text{ and } E_2$	<pre>{ E.pos:= CrearVarTemp() ;   emite( E.pos ':=' E1.pos '*' E2.pos ) }</pre>
$\rightarrow \text{not } E_1$	<pre>{ E.pos:= CrearVarTemp() ;   emite( E.pos ':=' 0 ) ;   emite( 'if' E1.pos '&lt;&gt; 0 goto' SIGINST+2 ) ;   emite( E.pos ':=' 1 ) }</pre>

15

## Instrucciones de control de flujo

$I \rightarrow \text{do } I_1 \text{ while } (E)$



$I \rightarrow \text{do}$	<pre>{ I.inicio := SIGINST }</pre>
$I_1$	
$\text{while } (E)$	<pre>{ emite( 'if' E.pos '&lt;&gt; 0 goto' I.inicio ) }</pre>


16



## Instrucciones de control de flujo

---

$I \rightarrow \text{while} ( E ) I$

$I \rightarrow \text{while}$	{ I.inicio:= SIGINST }	
( E )	{ emite( 'if' E.pos '=0 goto' I.fin ) }	
$I_1$	{ emite( 'goto' I.inicio ) ;	
	I.fin := SIGINST }	

17

---

## 6. Relleno por retroceso

18

## Perfil de las funciones usadas

---

### *ptro CreaLans(E)*

Crea una lista que solo contiene un número de instrucción E a rellenar posteriormente. Devuelve un puntero a dicha lista.

### *ptro CompletaLans( ptro, E )*

Rellena todas las instrucciones incompletas, cuyo número está contenido en la lista apuntada por ptro, con el valor de argumento E.

### *ptro FusionaLans (ptro, ptro)*

Concatena las listas apuntadas por sus dos argumentos y devuelve un puntero a la nueva lista.

19

---

## 7. Instrucciones de control de flujo

20

## While y if-else

$I \rightarrow \text{while}$ $(E)$	{ inicio:= SIGINST }
	{ I.final:= CreaLans(SIGINST); emite( 'if' E.pos '=0 goto' --- ) }
$I_1$	{ emite( 'goto' I.inicio ); CompletaLans(I.final, SIGINST) }

$I \rightarrow \text{if } E$	{ I.falso:= CreaLans(SIGINST); emite( 'if' E.pos '=0 goto' --- ); }
$I_1 \text{ else}$	{ I.fin := CreaLans( SIGINST ); emite( 'goto' --- ); CompletaLans(I.falso, SIGINST) }
$I_2$	{ CompletaLans (I.fin, SIGINST) }

21

## for

$I \rightarrow \text{for} ( I_1 ;$ $E ;$	{ I.cond := SIGINST; }
	{ I.fin := CreaLans (SIGINST); emite( 'if' E.pos '=0 goto' --- ); I.cuerpo := CreaLans (SIGINST); emite( 'goto' --- ); I.incr := SIGINST ; }
$I_2 )$	{ emite('goto' I.cond); CompletaLans(I.cuerpo, SIGINST) ; }
$I$	{ emite('goto' I.incr); CompletaLans(I.fin, SIGINST) ; }

22

## 8. Llamadas a subprogramas

23

Decl_Subprg $\rightarrow$ Tipo id ( Param_Form )	{ emite('push FP'); emite('FP := TOP'); area_datos := CreaLans(SIGINST); emite('TOP := TOP + ---') ; }
Bloque ;	{ CompletaLans (area_datos, DESP) emite('TOP := FP'); emite('FP := pop'); emite('ret') }

E $\rightarrow$ id ( Param_Act )	{ E.pos := CreaVarTemp(); emite('TOP := TOP + Talla_Tipo(id.nom)) } { emite('push' <estado máquina>; emite('call' BuscaPos(id.nom)) ; emite('pop' <estado máquina>; emite('TOP := TOP - TallaParam(id.nom)) ; emite('pop' E.pos) }
Param_Act $\rightarrow$ E	{ emite('push' E.pos) }
Param_Act $\rightarrow$ E , Param_Act	{ emite('push' E.pos) }