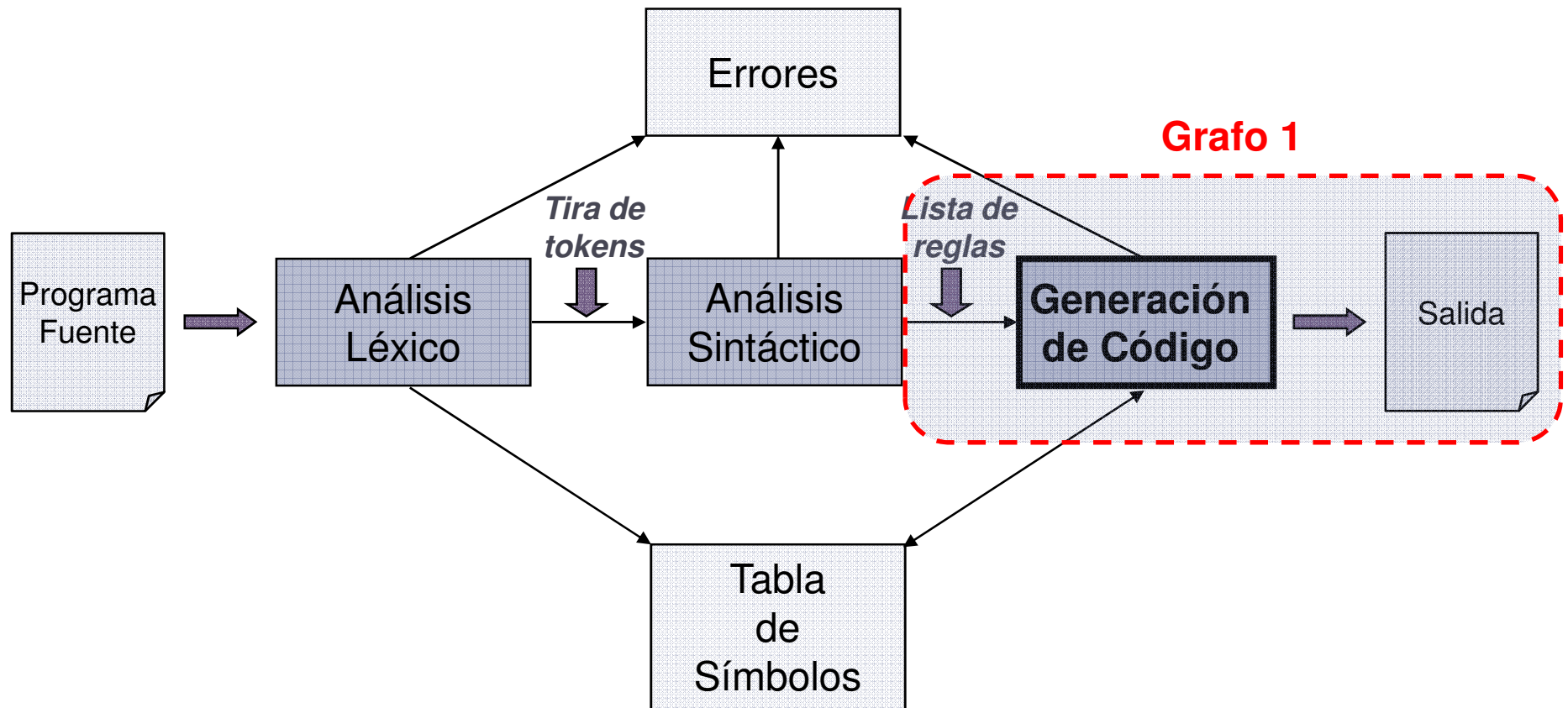


# Diseño de Compiladores I

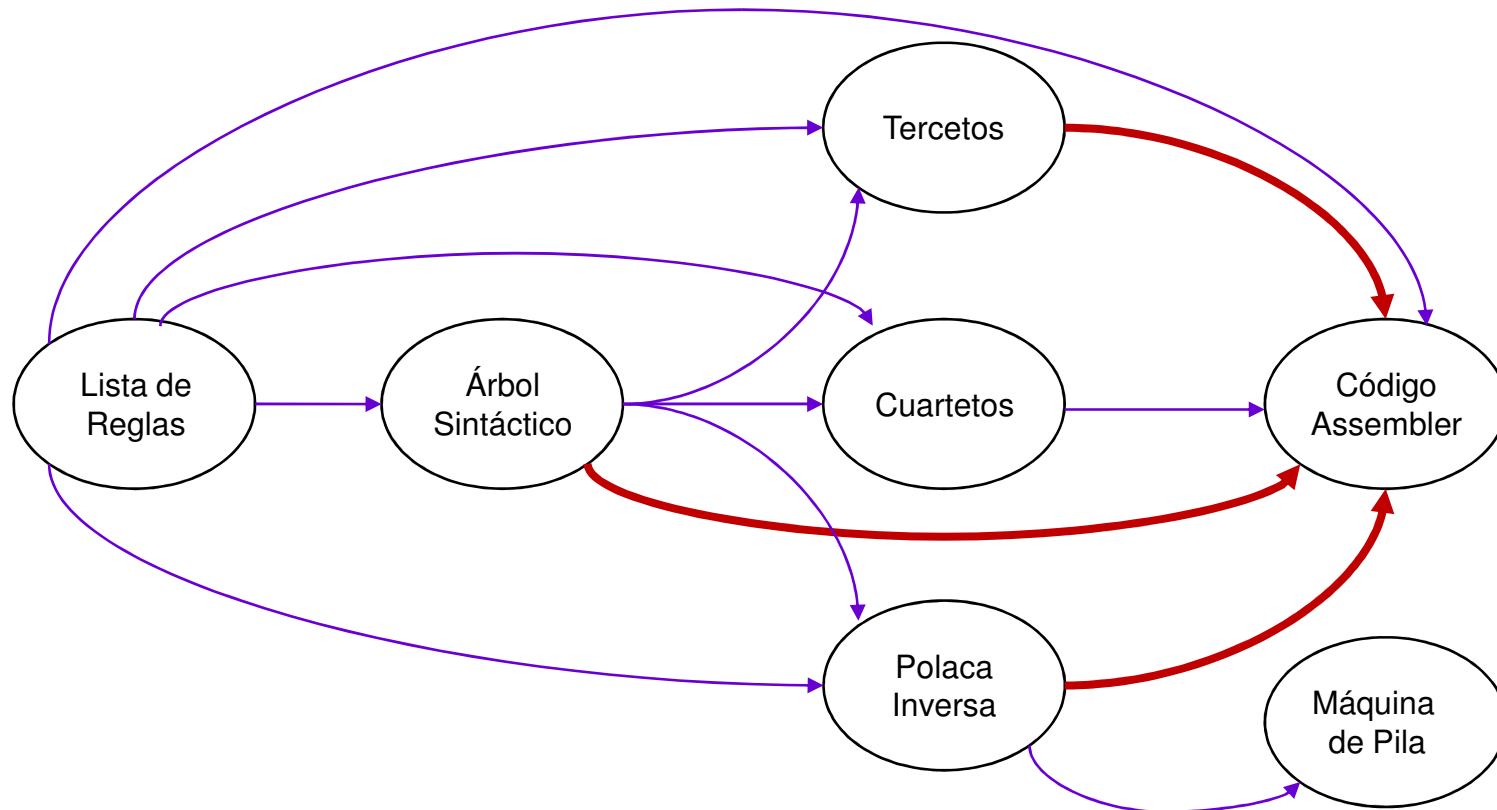
Generación de Código

# Fases de la Compilación



# Generación de Código

---



# ASIGNACIONES / EXPRESIONES

Tercetos → Assembler  
Polaca Inversa → Assembler  
Árbol Sintáctico → Assembler

Polaca Inversa → Assembler

# Polaca Inversa $\rightarrow$ Assembler

Ejemplo:  $z := a + b * c - d / (e + f) + 20$

Árbol Sintáctico  $\rightarrow$  Polaca Inversa

<b>z</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>*</b>	<b>+</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>+</b>	<b>/</b>	<b>-</b>	<b>20</b>	<b>+</b>	<b>:=</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	-----------

Lista de Reglas  $\rightarrow$  Polaca Inversa

<b>a</b>	<b>b</b>	<b>c</b>	<b>*</b>	<b>+</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>+</b>	<b>/</b>	<b>-</b>	<b>20</b>	<b>+</b>	<b>z</b>	<b>:=</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	----------	-----------



# Polaca Inversa → Assembler

## Variables Auxiliares

---

- ▶ Se apilan los operandos hasta llegar a un operador.
- ▶ **Operador binario:**
  - ▶ Desapilar 2 elementos
  - ▶ Generar código creando una variable auxiliar
  - ▶ Apilar variable auxiliar donde quedó el resultado
- ▶ **Operador unario:**
  - ▶ Desapilar 1 elemento
  - ▶ Generar código creando una variable auxiliar
  - ▶ Apilar variable auxiliar donde quedó el resultado



# Polaca Inversa → Assembler

## Variables Auxiliares

---

<b>z</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>*</b>	<b>+</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>+</b>	<b>/</b>	<b>-</b>	<b>20</b>	<b>+</b>	<b>:=</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	-----------

<b>Leído</b>	<b>Pila</b>
z	z
a	z a
b	z a b
c	z a b c
*	z a
<b>*</b>	<b>z a @aux1</b>

MOV RI, \_b

MUL RI, \_c

MOV @aux1, RI



# Variables Auxiliares

---

MOV RI, **\_b**

MUL RI, **\_c**

MOV **@aux1**, RI

- ▶ El texto marcado en rojo es texto fijo.
- ▶ Se agregan prefijos (**\_**, **@**) a las variables para evitar errores en la compilación del código Assembler.

Por ejemplo:

$x := MUL * k$

MOV RI, MUL ← Error de compilación

- ▶ Los prefijos para las variables auxiliares deben ser diferentes a los de las variables del usuario.
- ▶ Las variables auxiliares se guardan en la TdeS.

# Polaca Inversa → Assembler

## Variables Auxiliares

---

<b>z</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>*</b>	<b>+</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>+</b>	<b>/</b>	<b>-</b>	<b>20</b>	<b>+</b>	<b>:=</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	-----------

<b>Leído</b>	<b>Pila</b>
	z a @aux1
+	z
<b>+</b>	<b>z @aux2</b>

MOV R1, \_a

ADD R1, @aux1

MOV @aux2, R1



# Polaca Inversa → Assembler

## Variables Auxiliares

---

<b>z</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>*</b>	<b>+</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>+</b>	<b>/</b>	<b>-</b>	<b>20</b>	<b>+</b>	<b>:=</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	-----------

<b>Leído</b>	<b>Pila</b>
--------------	-------------

	z @aux2	MOV RI, _e
d	z @aux2 d	ADD RI, _f
e	z @aux2 d e	MOV @aux3, RI
f	z @aux2 d e f	
+	z @aux2 d	

<b>+</b>	<b>z @aux2 d @aux3</b>
----------	------------------------



# Polaca Inversa → Assembler

## Variables Auxiliares

---

z	a	b	c	*	+	d	e	f	+	/	-	20	+	:=
---	---	---	---	---	---	---	---	---	---	---	---	----	---	----

Leído	Pila
	z @aux2 d @aux3
/	z @aux2
/	z @aux2 @aux4

```
MOV RI, _d
DIV RI, @aux3
MOV @aux4, RI
```



# Polaca Inversa → Assembler

## Variables Auxiliares

---

z	a	b	c	*	+	d	e	f	+	/	-	20	+	:=
---	---	---	---	---	---	---	---	---	---	---	---	----	---	----

Leído	Pila
	z @aux2 @aux4
-	z
-	z @aux5

```
MOV R1, @aux2
SUB R1, @aux4
MOV @aux5, R1
```



# Polaca Inversa → Assembler

## Variables Auxiliares

---

z	a	b	c	*	+	d	e	f	+	/	-	20	+	:=
---	---	---	---	---	---	---	---	---	---	---	---	----	---	----

Leído	Pila
	z @aux5
20	z @aux5 20
+	z
+	z @aux6

```
MOV R1, @aux5
ADD R1, 20
MOV @aux6, R1
```



# Polaca Inversa → Assembler

## Variables Auxiliares

---

z	a	b	c	*	+	d	e	f	+	/	-	20	+	:=
---	---	---	---	---	---	---	---	---	---	---	---	----	---	----

Leído	Pila
	z @aux6
:=	-
:=	-

```
MOV R1, @aux6  
MOV _z, R1
```



# Variables Auxiliares

---

MOV R1, \_b

MUL R1, \_c

MOV @aux1, R1

MOV R1, \_a

ADD R1, @aux1

MOV @aux2, R1

MOV R1, \_e

ADD R1, \_f

MOV @aux3, R1

MOV R1, \_d

DIV R1, @aux3

MOV @aux4, R1

MOV R1, @aux2

SUB R1, @aux4

MOV @aux5, R1


MOV R1, @aux5

ADD R1, 20

MOV @aux6, R1

MOV R1, @aux6

MOV \_z, R1



20 instrucciones  
+  
6 variables  
auxiliares



# Polaca Inversa → Assembler

## Seguimiento de Registros

---

- ▶ Previo a la generación de código al encontrar un operador, se debe verificar qué registro está disponible.
- ▶ Se utiliza una tabla de ocupación de registros:

R1	R2	R3	R4
L	L	L	L

- ▶ Luego de generar código para los elementos desapilados, el registro que quedó ocupado por el resultado de la operación, quedará marcado como ocupado.
- ▶ Se apila el registro donde quedó el resultado.



# Polaca Inversa → Assembler

## Seguimiento de Registros

---

- ▶ Se apilan los operandos hasta llegar a un operador.
- ▶ **Operador binario:**
  - ▶ Desapilar 2 elementos
  - ▶ Generar código ocupando / liberando registros según corresponda
  - ▶ Apilar registro donde quedó el resultado
- ▶ **Operador unario:**
  - ▶ Desapilar 1 elemento
  - ▶ Generar código ocupando / liberando registros según corresponda
  - ▶ Apilar registro donde quedó el resultado



# Polaca Inversa → Assembler

## Seguimiento de Registros

<b>z</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>*</b>	<b>+</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>+</b>	<b>/</b>	<b>-</b>	<b>20</b>	<b>+</b>	<b>:=</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	-----------

Leído	Pila
z	z
a	z a
b	z a b
c	z a b c
*	z a
<b>*</b>	<b>z a RI</b>

RI	R2	R3	R4
⊥	L	L	L
0			

```
MOV RI, _b
MUL RI, _c
```



# Polaca Inversa → Assembler

## Seguimiento de Registros

---

<b>z</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>*</b>	<b>+</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>+</b>	<b>/</b>	<b>-</b>	<b>20</b>	<b>+</b>	<b>:=</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	-----------

<b>Leído</b>	<b>Pila</b>
	z a RI
+	z
<b>+</b>	<b>z RI</b>

<b>RI</b>	<b>R2</b>	<b>R3</b>	<b>R4</b>
Ł	L	L	L
O			

ADD RI, \_a



# Polaca Inversa → Assembler

## Seguimiento de Registros

<b>z</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>*</b>	<b>+</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>+</b>	<b>/</b>	<b>-</b>	<b>20</b>	<b>+</b>	<b>:=</b>
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	-----------

Leído	Pila
-------	------

	z RI
d	z RI d
e	z RI d e
f	z RI d e f
+	z RI d

<b>+</b>	<b>z RI d R2</b>
----------	------------------

RI	R2	R3	R4
⊥	⊥	L	L
0	0		

```
MOV R2, _e
ADD R2, _f
```



# Polaca Inversa → Assembler

## Seguimiento de Registros

---

z	a	b	c	*	+	d	e	f	+	/	-	20	+	:=
---	---	---	---	---	---	---	---	---	---	---	---	----	---	----

Leído	Pila
	z R1 d R2
/	z R1
/	<b>z R1 R3</b>

R1	R2	R3	R4
⊥	L	⊥	L
0		0	

```
MOV R3, _d
DIV R3, R2
```



# Polaca Inversa → Assembler

## Seguimiento de Registros

---

z	a	b	c	*	+	d	e	f	+	/	-	20	+	:=
---	---	---	---	---	---	---	---	---	---	---	---	----	---	----

Leído	Pila
	z RI R3
-	z
-	z RI

RI	R2	R3	R4
⊥	L	L	L
0			

SUB RI, R3



# Polaca Inversa → Assembler

## Seguimiento de Registros

---

z	a	b	c	*	+	d	e	f	+	/	-	20	+	:=
---	---	---	---	---	---	---	---	---	---	---	---	----	---	----

Leído	Pila
	z RI
20	z RI 20
+	z
+	z RI

RI	R2	R3	R4
Ł	L	L	L
O			

ADD RI, 20





# Polaca Inversa → Assembler

## Seguimiento de Registros

---

z	a	b	c	*	+	d	e	f	+	/	-	20	+	:=
---	---	---	---	---	---	---	---	---	---	---	---	----	---	----

Leído	Pila
	z RI
:=	-
:=	-

RI	R2	R3	R4
L	L	L	L

MOV \_z, RI



# Polaca Inversa → Assembler

## Seguimiento de Registros

---

```
MOV R1, _b
MUL R1, _c
ADD R1, _a
MOV R2, _e
ADD R2, _f
MOV R3, _d
DIV R3, R2
SUB R1, R3
ADD R1, 20
MOV _z, R1
```

10 instrucciones  
Y  
ninguna variable  
auxiliar



Código más chico



Código más rápido



# Polaca Inversa → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

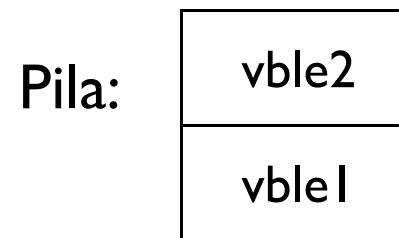
---

### Situación 1:

Operación entre 2 variables y/o constantes



1. Ocupar un registro
2. Generar código sobre ese registro



R1	R2	R3	R4
L	L	L	L
O			

MOV R1, vble1

OP R1, vble2



Generación de Código

# Polaca Inversa → Assembler

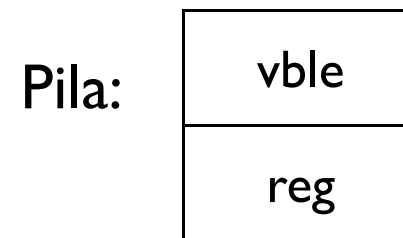
## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 2:

Operación entre un registro y una variable o constante



1. Generar código sobre ese registro  
(Tabla sin cambios)

OP RI, vble

RI	R2	R3	R4
O	L	L	L

---

Generación de Código

# Polaca Inversa → Assembler

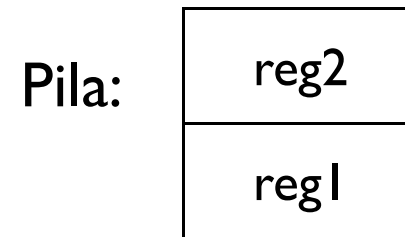
## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 3:

Operación entre dos registros



1. Generar código sobre el primer registro
2. Liberar el segundo registro

OP R1, R2

R1	R2	R3	R4
O	⊖ L	L	L

---

Generación de Código

# Polaca Inversa → Assembler

## Seguimiento de Registros

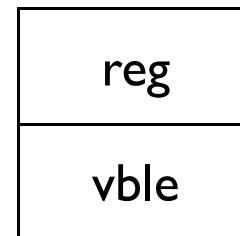
Decisiones del Algoritmo:

---

### Situación 4:

Operación entre una variable (o constante) y un registro

Pila:



# Polaca Inversa → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 4.a:

Operación conmutativa entre una variable (o constante) y un registro

vble	reg	+
------	-----	---

1. Generar código sobre el registro  
(Tabla sin cambios)

Pila:

reg
vble

ADD R1, vble

R1	R2	R3	R4
0	L	L	L

---

Generación de Código

# Polaca Inversa → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

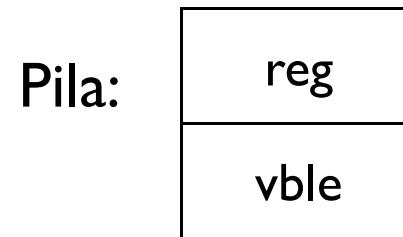
---

### Situación 4.b:

Operación no conmutativa entre una variable (o constante) y un registro

vble	reg	/
------	-----	---

1. Ocupar nuevo registro
2. Generar código sobre el nuevo registro
3. Liberar el primer registro



R1	R2	R3	R4
○	⊥ ○	L	L

MOV R2, vble

DIV R2, R1

R1	R2	R3	R4
⊖ L	○	L	L





# Polaca Inversa → Assembler

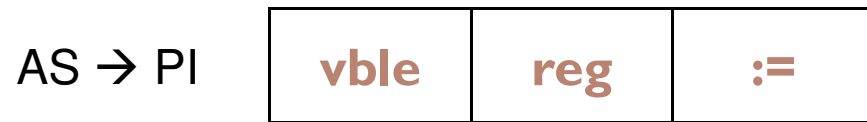
## Seguimiento de Registros

Decisiones del Algoritmo:

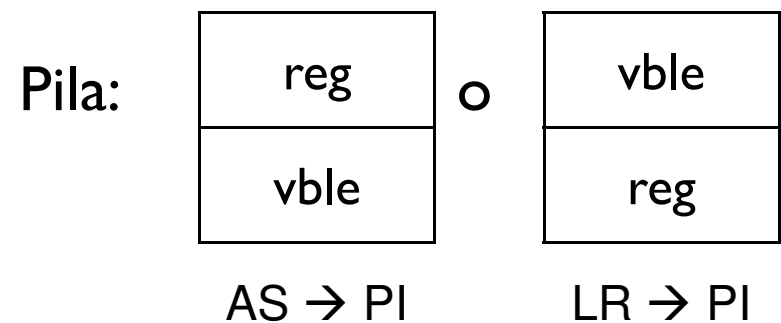
---

### Asignaciones – Situación a:

Valor a asignar en un registro



o



MOV vble, R1

R1	R2	R3	R4
⊖ L	L	L	L

1. Generar código usando el registro
2. Liberar el registro



Generación de Código

# Polaca Inversa → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

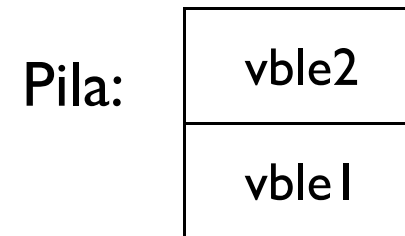
---

### Asignaciones – Situación b:

Valor a asignar en una variable (o constante)

vble1	vble2	:=
-------	-------	----

1. Ocupar registro
2. Generar código usando el registro
3. Liberar el registro



R1	R2	R3	R4
⊥ O	L	L	L

MOV R1, vble2

MOV vble1, R1

R1	R2	R3	R4
⊖ L	L	L	L

---

Generación de Código

# Seguimiento de Registros

## Para cualquier representación intermedia

---

- ▶ El **número máximo de registros** necesarios depende del número de niveles de precedencia:
  - ▶ Con 2 niveles ( + y -, \* y / ) → 2 registros
  - ▶ Con 3 niveles ( potencia ) → 3 registros
  - ▶ Con 4 niveles ( < y > ) → 4 registros
  - ▶ Con 5 niveles ( AND y OR ) → 5 registros



# Seguimiento de Registros

---

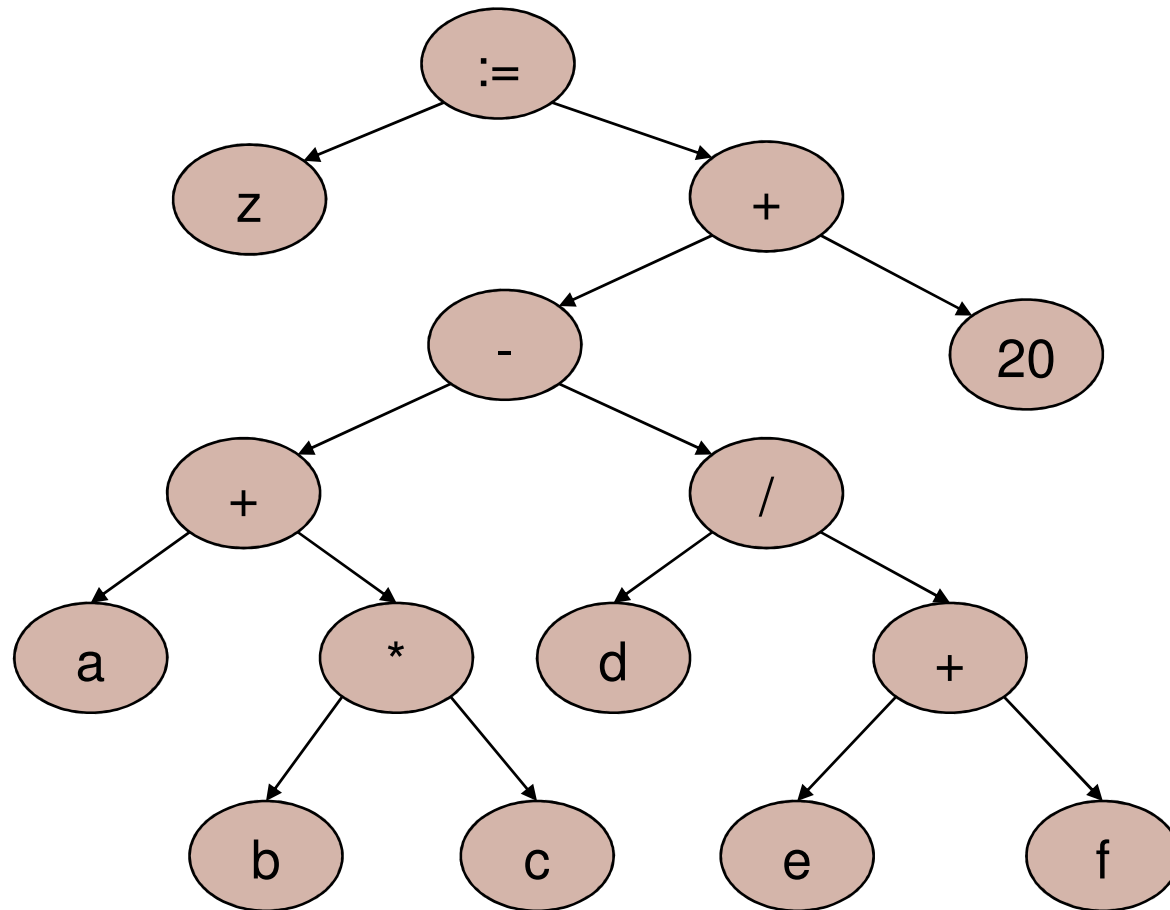
- ▶ Si se necesita un registro y no hay más, se libera un registro, moviendo su contenido a una variable auxiliar.
- ▶ Se posterga el uso de registros que tienen usos específicos (sirven para más operaciones).
- ▶ Ejemplo:  
Registros para Pentium: EAX, EBX, ECX, EDX
- ▶ EAX y EDX se utilizan para multiplicaciones y divisiones
  - Comenzar utilizando los otros 2.



# Árbol Sintáctico → Assembler

# Árbol Sintáctico → Assembler

Ejemplo:  $z := a + b * c - d / (e + f) + 20.$



# Árbol Sintáctico → Assembler

## Variables Auxiliares

---

- ▶ Se busca el subárbol de más a la izquierda con hijos hojas.

(Si se trata de un operador unario, será un subárbol con un solo hijo hoja)

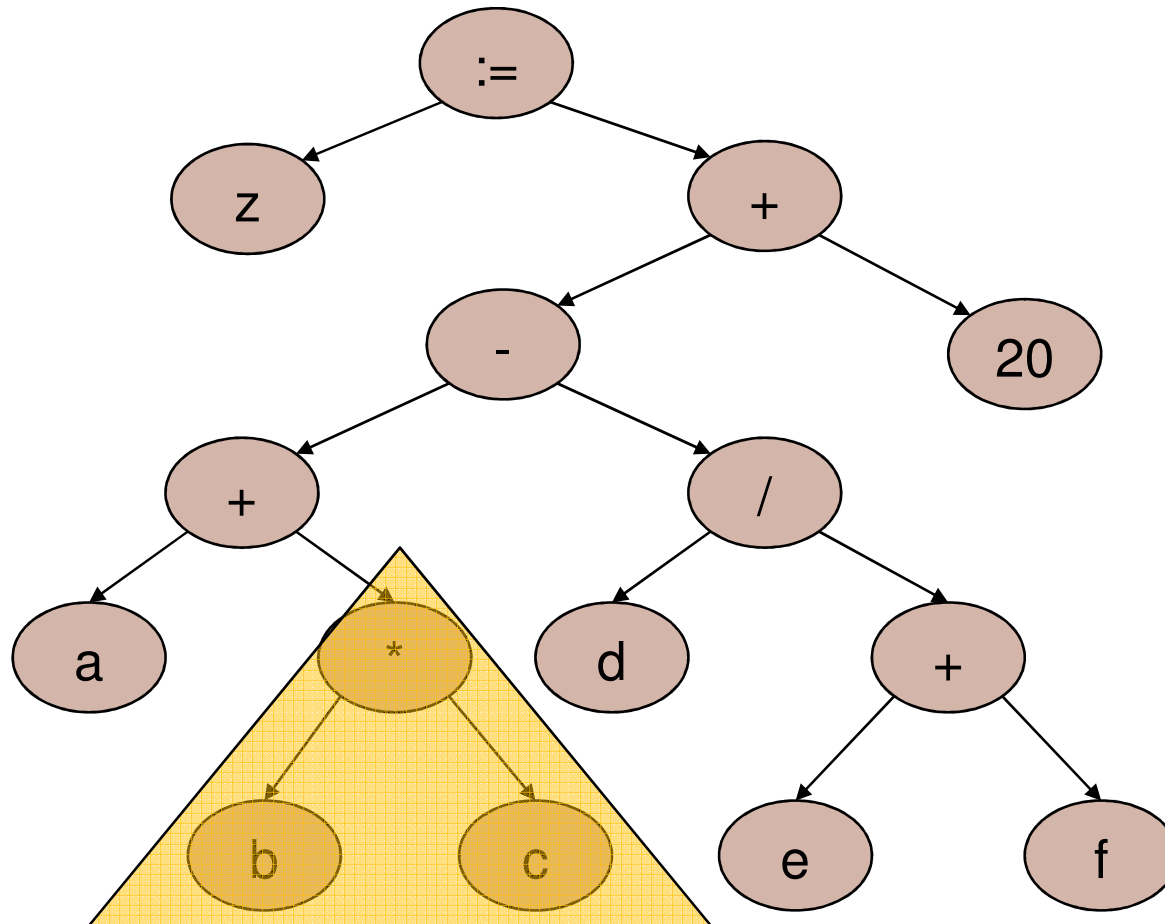
- ▶ Se genera código para el subárbol, creando una variable auxiliar
- ▶ Se reemplaza el subárbol por la variable auxiliar donde quedó el resultado de la operación.



# Árbol Sintáctico → Assembler

## Variables Auxiliares

---



```
MOV RI, _b  
MUL RI, _c  
MOV @auxI, RI
```





# Árbol Sintáctico → Assembler

## Variables Auxiliares

---

MOV RI, \_b

MUL RI, \_c

MOV @aux1, RI

- ▶ El texto marcado en rojo es texto fijo.
- ▶ Se agregan prefijos (\_, @) a las variables para evitar errores en la compilación del código Assembler.

Por ejemplo:

$x := MUL * k$

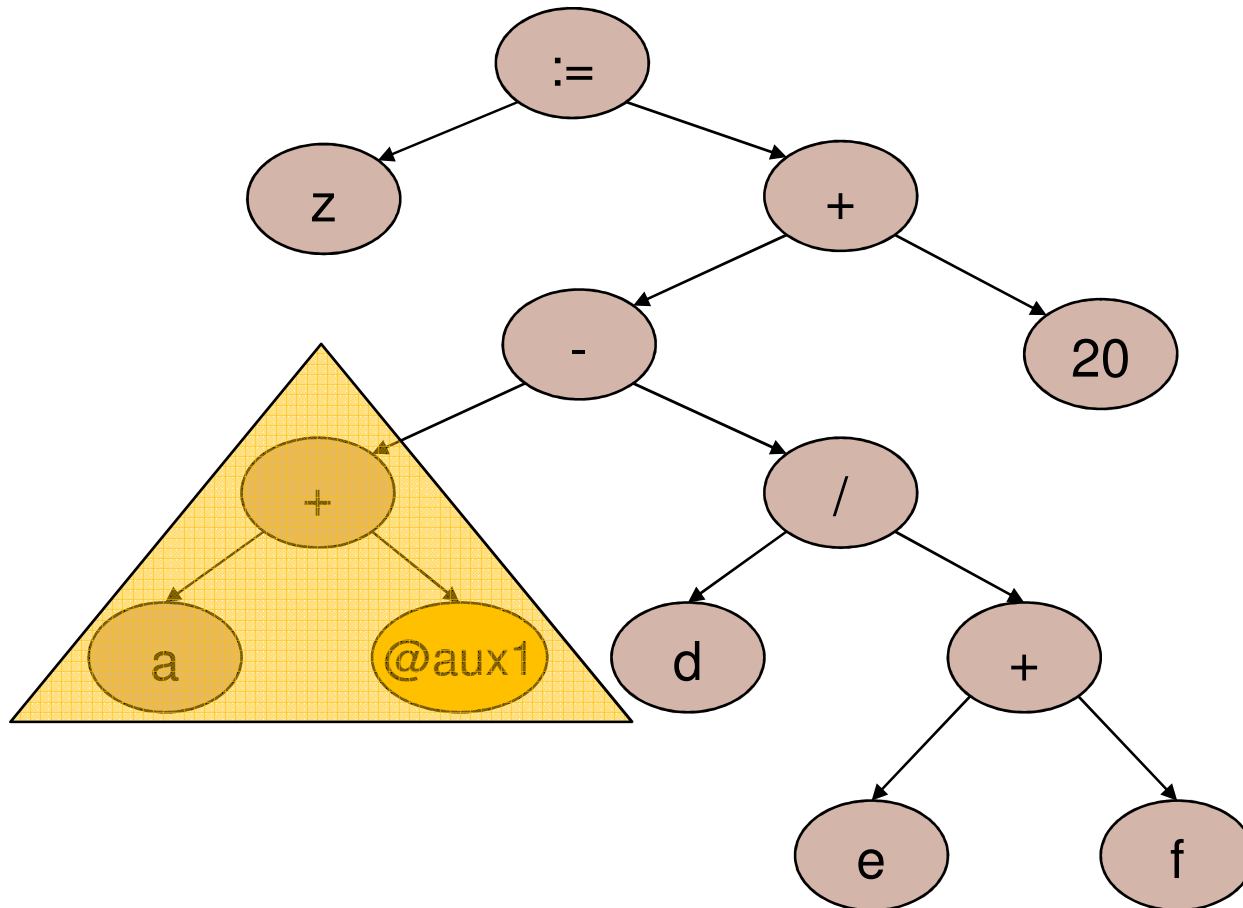
MOV RI, MUL ← Error de compilación

- ▶ Los prefijos para las variables auxiliares deben ser diferentes a los de las variables del usuario.
- ▶ Las variables auxiliares se guardan en la TdeS.

# Árbol Sintáctico → Assembler

## Variables Auxiliares

---



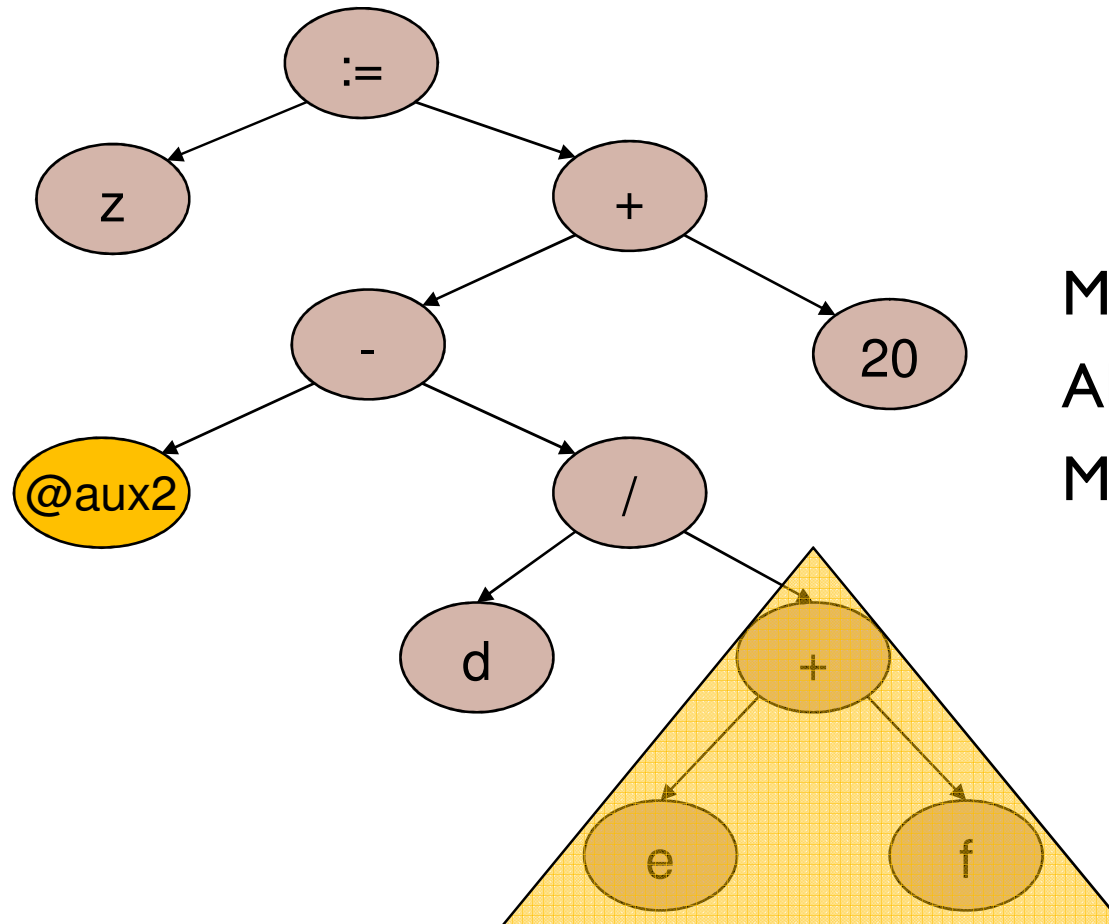
```
MOV RI, _a  
ADD RI, @aux1  
MOV @aux2, RI
```



# Árbol Sintáctico → Assembler

## Variables Auxiliares

---



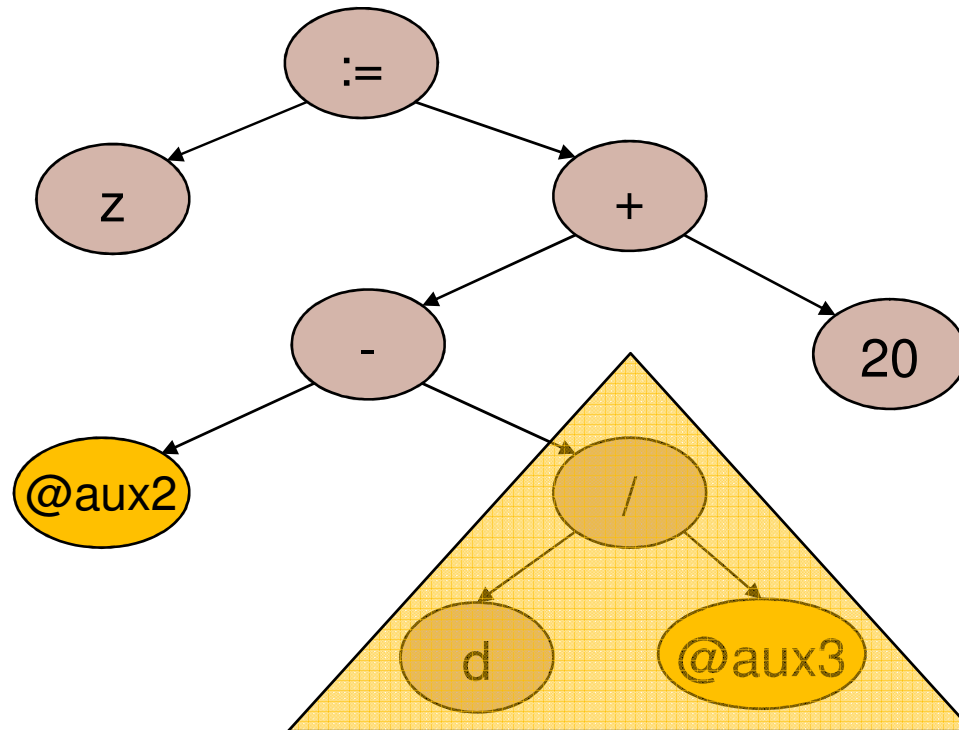
```
MOV RI, _e  
ADD RI, _f  
MOV @aux3, RI
```



# Árbol Sintáctico → Assembler

## Variables Auxiliares

---



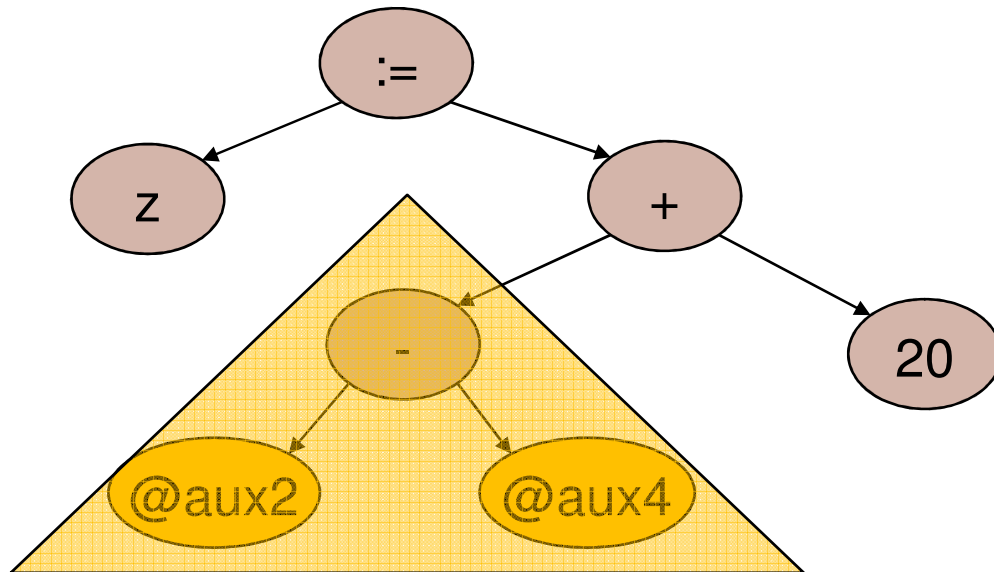
```
MOV RI, _d  
DIV RI, @aux3  
MOV @aux4, RI
```



# Árbol Sintáctico → Assembler

## Variables Auxiliares

---



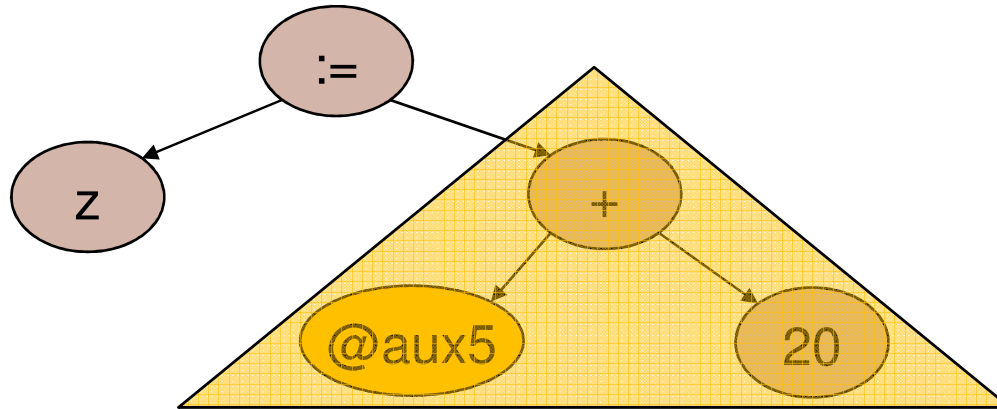
```
MOV RI, @aux2  
SUB RI, @aux4  
MOV @aux5, RI
```



# Árbol Sintáctico → Assembler

## Variables Auxiliares

---



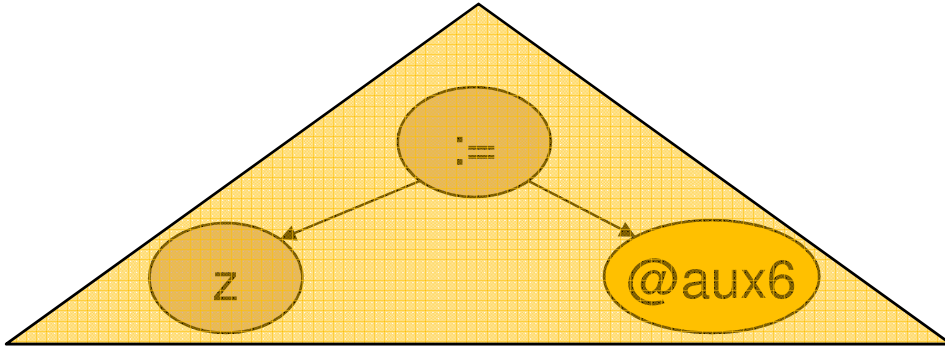
```
MOV R1, @aux5  
ADD R1, 20  
MOV @aux6, R1
```



# Árbol Sintáctico → Assembler

## Variables Auxiliares

---



```
MOV R1, @aux6  
MOV _z, R1
```



# Árbol Sintáctico → Assembler

## Variables Auxiliares

---

MOV RI, \_b

MUL RI, \_c

MOV @aux1, RI

MOV RI, \_a

ADD RI, @aux1

MOV @aux2, RI

MOV RI, \_e

ADD RI, \_f

MOV @aux3, RI

MOV RI, \_d

DIV RI, @aux3

MOV @aux4, RI

MOV RI, @aux2

SUB RI, @aux4

MOV @aux5, RI

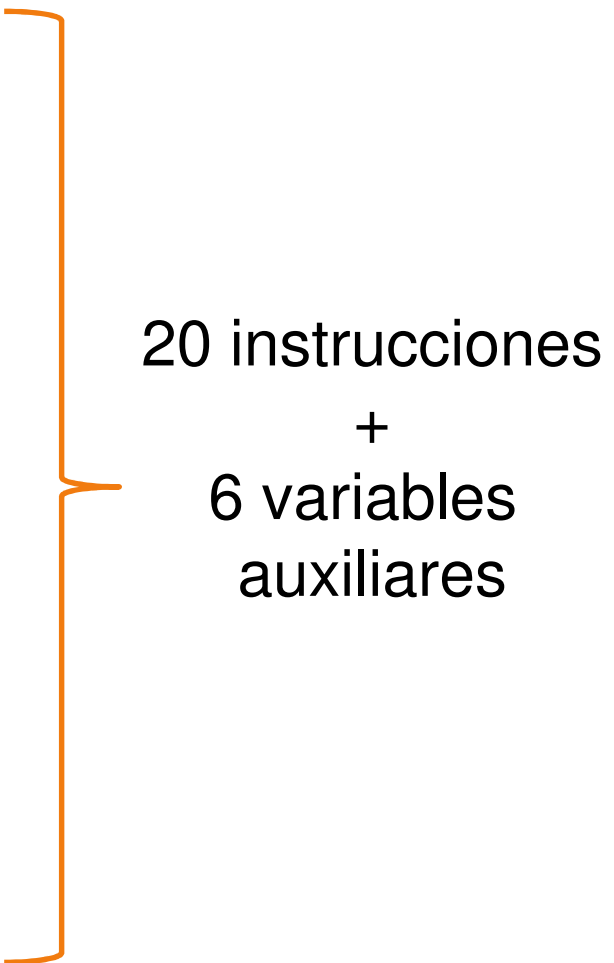
MOV RI, @aux5

ADD RI, 20

MOV @aux6, RI

MOV RI, @aux6

MOV \_z, RI



20 instrucciones  
+  
6 variables  
auxiliares



# Árbol Sintáctico → Assembler

## Seguimiento de Registros

---

- ▶ Previo a la generación de código para un subárbol, se debe verificar qué registro está disponible.
- ▶ Se utiliza una tabla de ocupación de registros:

R1	R2	R3	R4
L	L	L	L

- ▶ Luego de generar código para el subárbol, se marcará el registro que quedó ocupado por el resultado de la operación.
- ▶ Se reemplaza el subárbol por el registro donde quedó el resultado.



# Árbol Sintáctico → Assembler

## Seguimiento de Registros

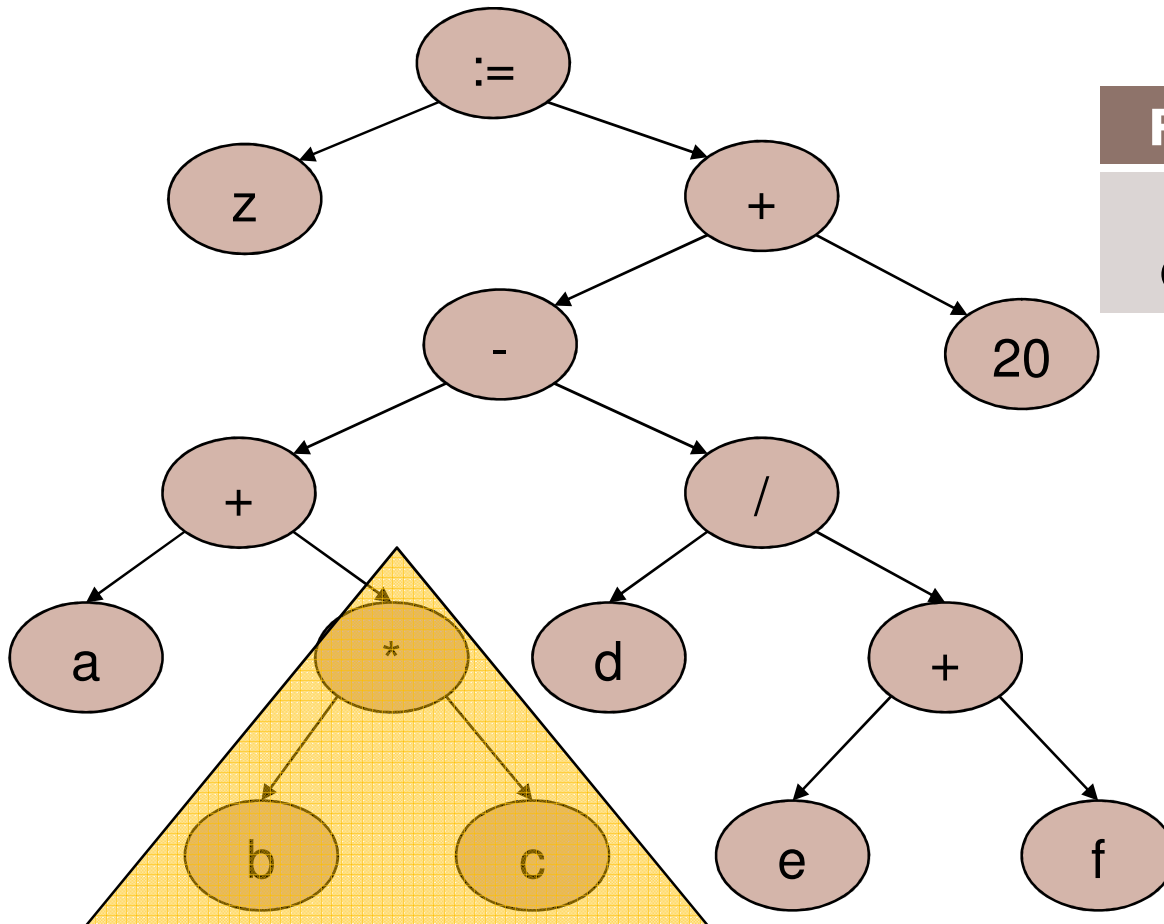
---

- ▶ Se busca el subárbol de más a la izquierda con hijos hojas.  
(Si se trata de un operador unario, será un subárbol con un solo hijo hoja)
- ▶ Se genera código para el subárbol, ocupando/liberando registros según corresponda.
- ▶ Se reemplaza el subárbol por el registro donde quedó el resultado de la operación.



# Árbol Sintáctico → Assembler

## Seguimiento de Registros



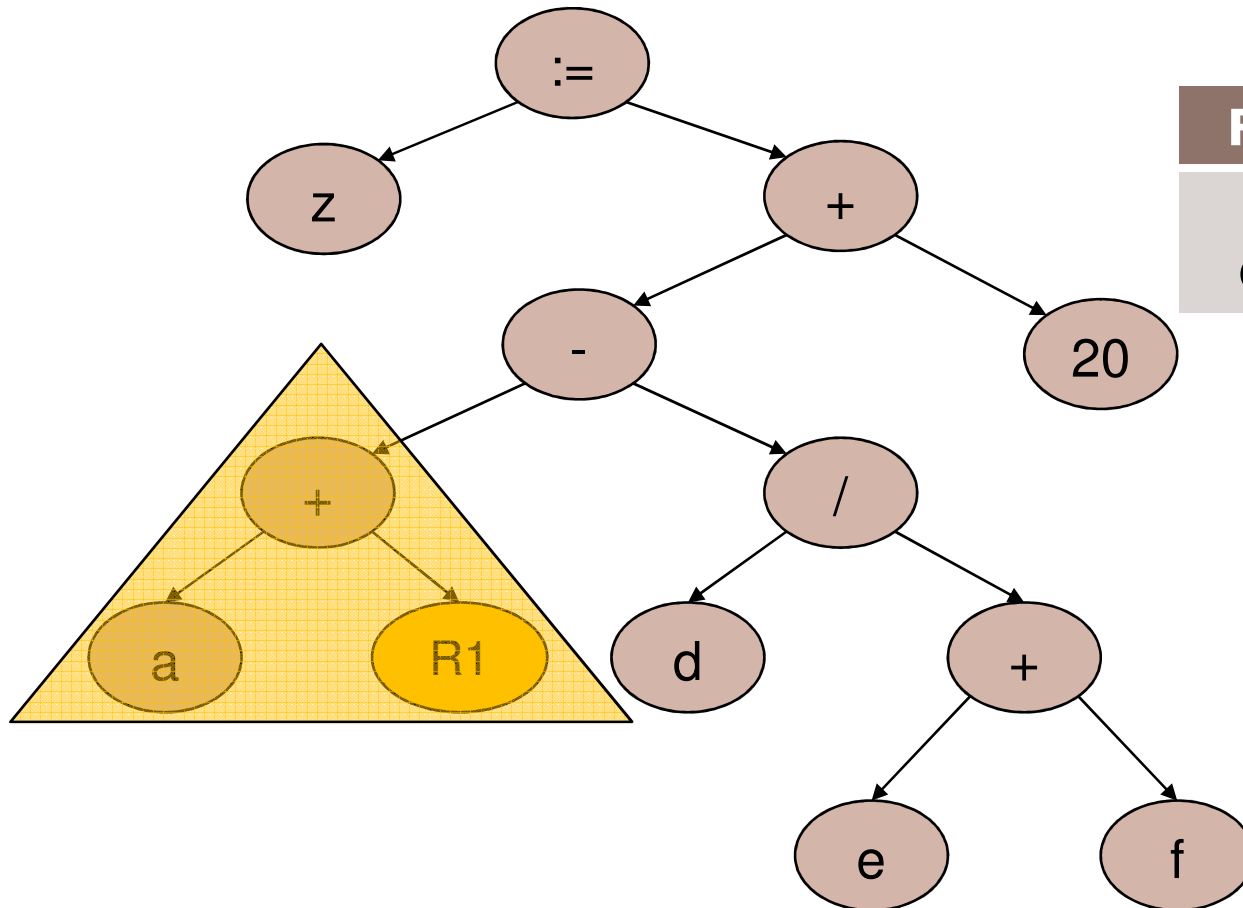
RI	R2	R3	R4
L	L	L	L
O			

MOV RI, \_b  
MUL RI, \_c



# Árbol Sintáctico → Assembler

## Seguimiento de Registros



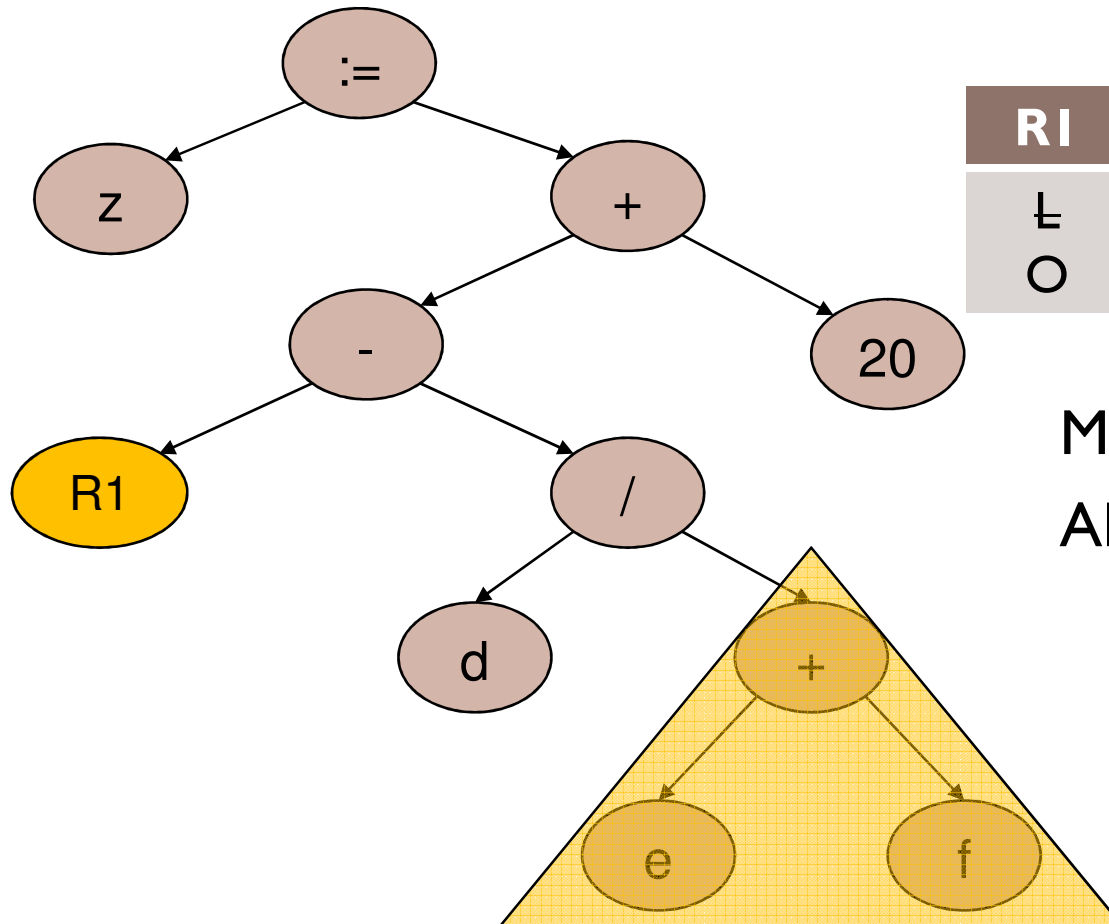
R1	R2	R3	R4
L	L	L	L
O			

ADD R1, \_a



# Árbol Sintáctico → Assembler

## Seguimiento de Registros



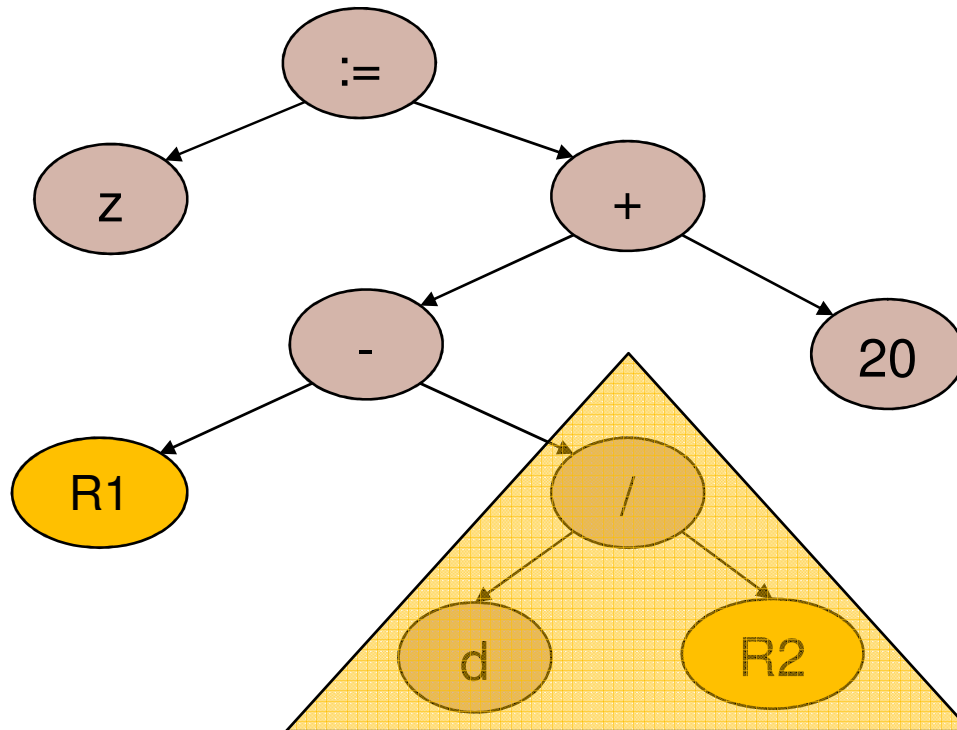
R1	R2	R3	R4
⊥	⊥	L	L
O	O		

MOV R2, \_e  
ADD R2, \_f



# Árbol Sintáctico → Assembler

## Seguimiento de Registros



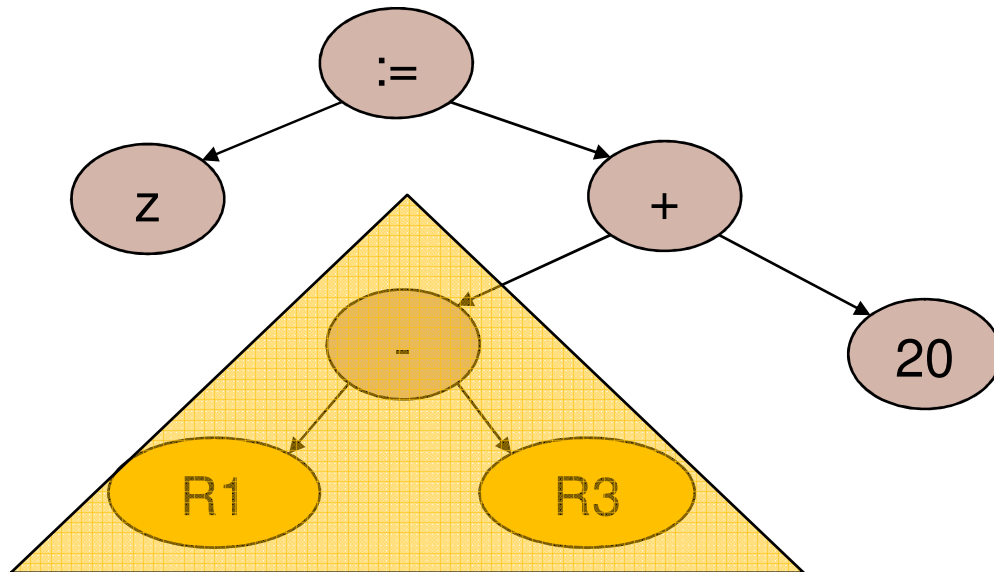
R1	R2	R3	R4
LO	L	LO	L

MOV R3, \_d  
DIV R3, R2



# Árbol Sintáctico → Assembler

## Seguimiento de Registros



RI	R2	R3	R4
L	L	L	L
O			

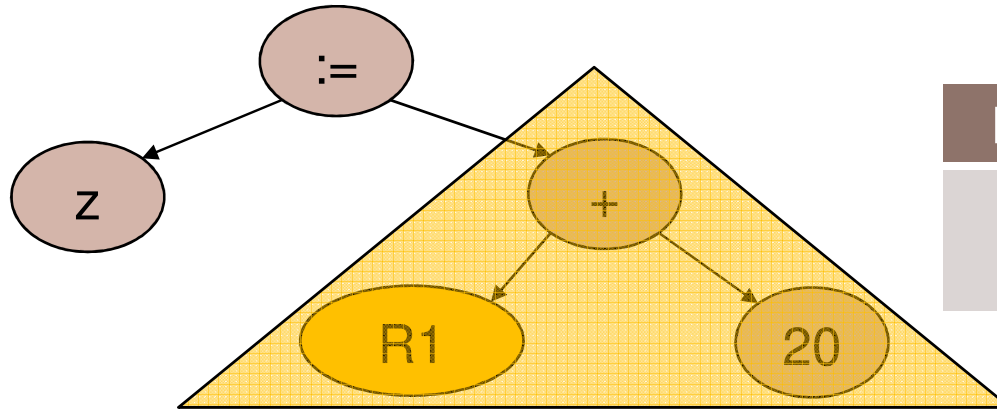
**SUB RI, R3**



# Árbol Sintáctico → Assembler

## Seguimiento de Registros

---



RI	R2	R3	R4
L O	L	L	L

**ADD RI, 20**

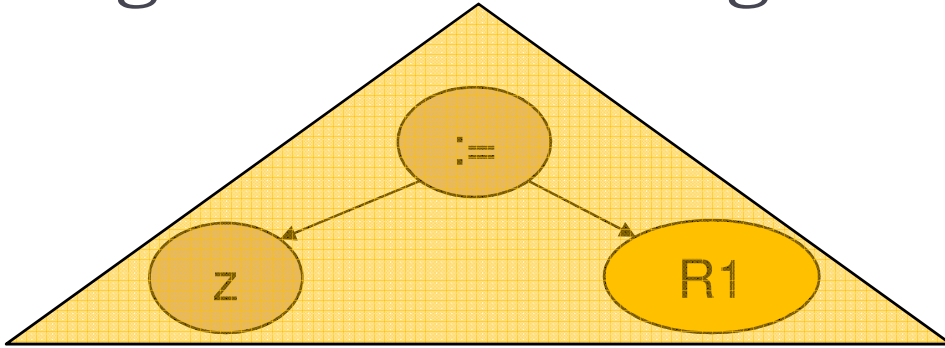




# Árbol Sintáctico → Assembler

## Seguimiento de Registros

---



RI	R2	R3	R4
L	L	L	L

MOV \_z, R1



# Árbol Sintáctico → Assembler

## Seguimiento de Registros

---

```
MOV R1, _b
MUL R1, _c
ADD R1, _a
MOV R2, _e
ADD R2, _f
MOV R3, _d
DIV R3, R2
SUB R1, R3
ADD R1, 20
MOV _z, R1
```

10 instrucciones  
Y  
ninguna variable  
auxiliar



Código más chico



Código más rápido



# Arbol Sintáctico → Assembler

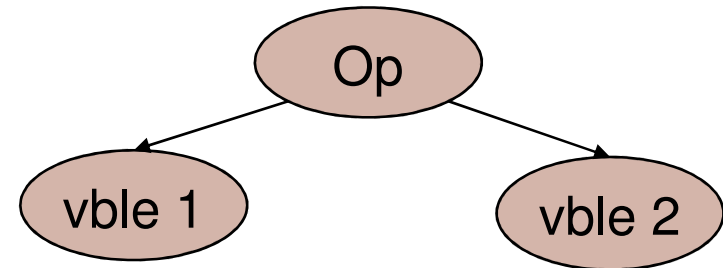
## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 1:

Operación entre 2 variables y/o constantes



1. Ocupar un registro
2. Generar código sobre ese registro

R1	R2	R3	R4
⊥	L	L	L
○			

MOV R1, vble1

OP R1, vble2



# Arbol Sintáctico → Assembler

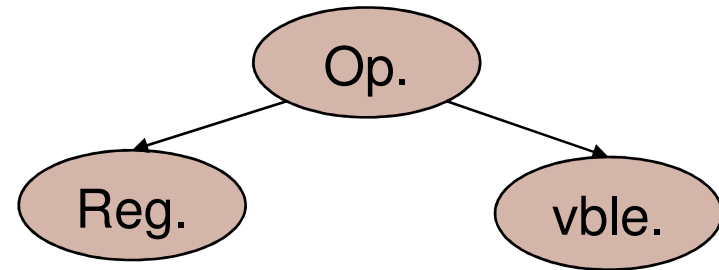
## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 2:

Operación entre un registro y una variable o constante



1. Generar código sobre ese registro  
(Tabla sin cambios)

OP RI, vble

RI	R2	R3	R4
O	L	L	L



Generación de Código

# Arbol Sintáctico → Assembler

## Seguimiento de Registros

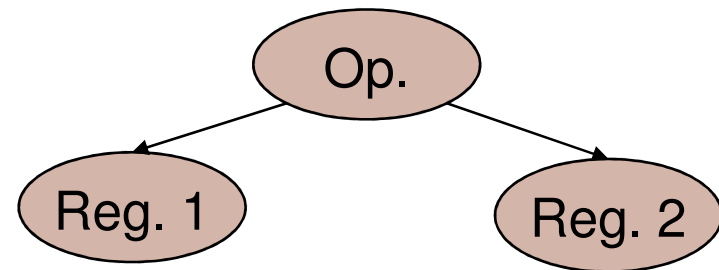
Decisiones del Algoritmo:

---

### Situación 3:

Operación entre dos registros

1. Generar código sobre el primer registro
2. Liberar el segundo registro



OP R1, R2

R1	R2	R3	R4
O	⊖ L	L	L

---

► Generación de Código

# Arbol Sintáctico → Assembler

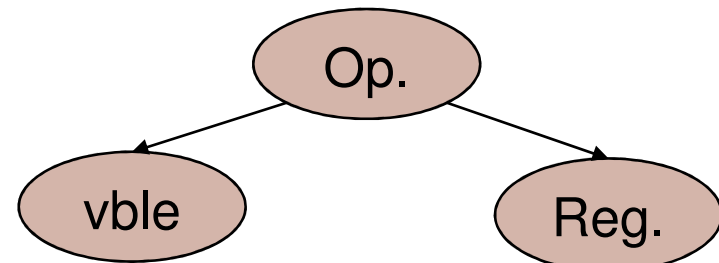
## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 4:

Operación entre una variable (o constante) y un registro



Generación de Código

# Arbol Sintáctico → Assembler

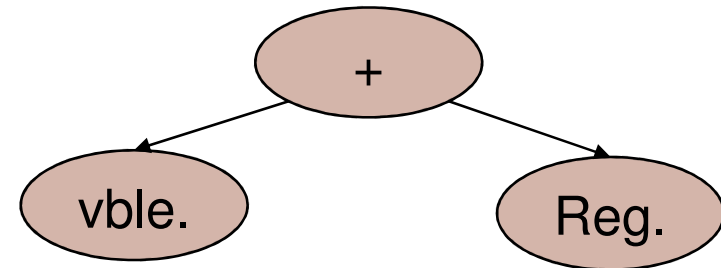
## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 4.a:

Operación conmutativa entre una variable (o constante) y un registro



1. Generar código sobre el registro

(Tabla sin cambios)

ADD R1, vble

R1	R2	R3	R4
0	L	L	L



# Arbol Sintáctico → Assembler

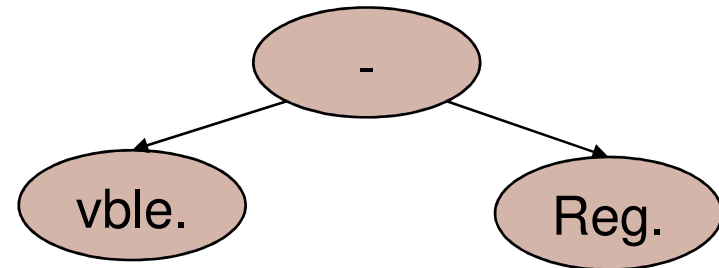
## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 4.b:

Operación no conmutativa entre una variable (o constante) y un registro



1. Ocupar nuevo registro
2. Generar código sobre el nuevo registro
3. Liberar el primer registro

RI	R2	R3	R4
○	⊥ ○	L	L

MOV R2, vble

SUB R2, RI

RI	R2	R3	R4
⊖ L	○	L	L





# Arbol Sintáctico → Assembler

## Seguimiento de Registros

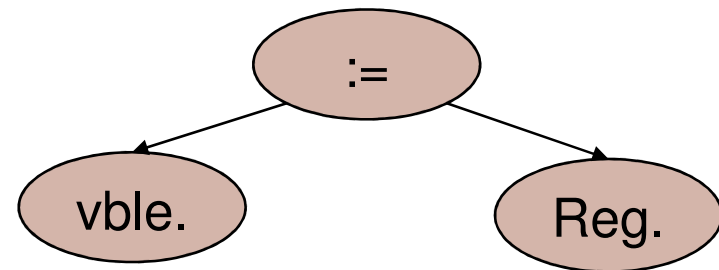
Decisiones del Algoritmo:

---

### Asignaciones – Situación a:

Valor a asignar en un registro

1. Generar código usando el registro
2. Liberar el registro



MOV vble, R1

R1	R2	R3	R4
⊖ L	L	L	L



Generación de Código

# Arbol Sintáctico → Assembler

## Seguimiento de Registros

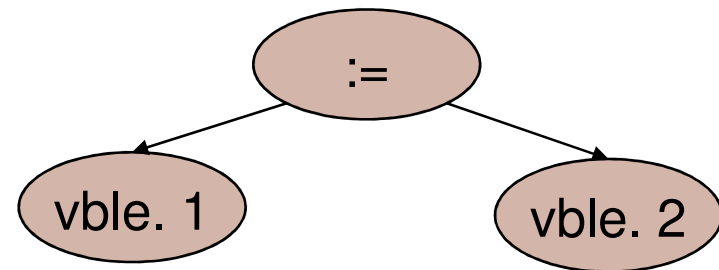
Decisiones del Algoritmo:

---

### Asignaciones – Situación b:

Valor a asignar en una variable (o constante)

1. Ocupar registro
2. Generar código usando el registro
3. Liberar el registro



R1	R2	R3	R4
⊥	L	L	L
0			

MOV R1, vble2

MOV vble1, R1

R1	R2	R3	R4
⊕	L	L	L
L			

---

Generación de Código

Tercetos → Assembler

# Tercetos $\rightarrow$ Assembler

---

Ejemplo:  $z := a + b * c - d / (e + f) + 20$

10. ( \* , b , c )
11. ( + , a , [10] )
12. ( + , e , f )
13. ( / , d , [12] )
14. ( - , [11] , [13] )
15. ( + , [14] , 20 )
16. ( := , z , [15] )



# Tercetos → Assembler

## Variables Auxiliares

---

- ▶ Se genera código para cada terceto, creando una variable auxiliar para almacenar el resultado.
- ▶ Se agrega al terceto, la información de la variable utilizada.



# Tercetos → Assembler

## Variables Auxiliares

---

10. ( \* , b , c ) @aux1

11. ( + , a , [10] )

12. ( + , e , f )

13. ( / , d , [12] )

14. ( - , [11] , [13] )

15. ( + , [14] , 20 )

16. ( := , z , [15] )

MOV RI, \_b

MUL RI, \_c

MOV @aux1, RI



# Variables Auxiliares

---

MOV RI, **\_b**

MUL RI, **\_c**

MOV **@aux1**, RI

- ▶ El texto marcado en rojo es texto fijo.
- ▶ Se agregan prefijos (**\_**, **@**) a las variables para evitar errores en la compilación del código Assembler.

Por ejemplo:

$x := MUL * k$

MOV RI, MUL ← Error de compilación

- ▶ Los prefijos para las variables auxiliares deben ser diferentes a los de las variables del usuario.
- ▶ Las variables auxiliares se guardan en la TdeS.

# Tercetos → Assembler

## Variables Auxiliares

---

10. ( \* , b , c ) @aux1

11. ( + , a , [10] ) @aux2

12. ( + , e , f )

13. ( / , d , [12] )

14. ( - , [11] , [13] )

15. ( + , [14] , 20 )

16. ( := , z , [15] )

MOV R1, \_a

ADD R1, @aux1

MOV @aux2, R1





# Tercetos → Assembler

## Variables Auxiliares

---

10. ( \* , b , c ) @aux1
11. ( + , a , [10] ) @aux2
12. ( + , e , f ) @aux3
13. ( / , d , [12] )
14. ( - , [11] , [13] )
15. ( + , [14] , 20 )
16. ( := , z , [15] )

```
MOV R1, _e
ADD R1, _f
MOV @aux3, R1
```



# Tercetos → Assembler

## Variables Auxiliares

---

- 10. ( \* , b , c ) @aux1
- 11. ( + , a , [10] ) @aux2
- 12. ( + , e , f ) @aux3
- 13. ( / , d , [12] ) @aux4
- 14. ( - , [11] , [13] )
- 15. ( + , [14] , 20 )
- 16. ( := , z , [15] )

```
MOV RI, _d
DIV RI, @aux3
MOV @aux4, RI
```



# Tercetos → Assembler

## Variables Auxiliares

---

10. ( \* , b , c ) @aux1

11. ( + , a , [10] ) @aux2

12. ( + , e , f ) @aux3

13. ( / , d , [12] ) @aux4

14. ( - , [11] , [13] ) @aux5

15. ( + , [14] , 20 )

16. ( := , z , [15] )

MOV R1, @aux2

SUB R1, @aux4

MOV @aux5, R1



# Tercetos → Assembler

## Variables Auxiliares

---

10. ( \* , b , c ) @aux1
11. ( + , a , [10] ) @aux2
12. ( + , e , f ) @aux3
13. ( / , d , [12] ) @aux4
14. ( - , [11] , [13] ) @aux5
15. ( + , [14] , 20 ) @aux6
16. ( := , z , [15] )

```
MOV R1, @aux5  
ADD R1, 20  
MOV @aux6, R1
```



# Tercetos → Assembler

## Variables Auxiliares

---

10. ( \* , b , c ) @aux1
11. ( + , a , [10] ) @aux2
12. ( + , e , f ) @aux3
13. ( / , d , [12] ) @aux4
14. ( - , [11] , [13] ) @aux5
15. ( + , [14] , 20 ) @aux6
16. ( := , z , [15] )

```
MOV R1, @aux6  
MOV _z, R1
```




# Variables Auxiliares

---

```
MOV RI, _b
MUL RI, _c
MOV @aux1, RI
MOV RI, _a
ADD RI, @aux1
MOV @aux2, RI
MOV RI, _e
ADD RI, _f
MOV @aux3, RI
MOV RI, _d
```

```
DIV RI, @aux3
MOV @aux4, RI
MOV RI, @aux2
SUB RI, @aux4
MOV @aux5, RI
MOV RI, @aux5
ADD RI, 20
MOV @aux6, RI
MOV RI, @aux6
MOV _z, RI
```



20 instrucciones  
+  
6 variables  
auxiliares

# Tercetos → Assembler

## Seguimiento de Registros

---

- ▶ Previo a la generación de código para un terceto, se debe verificar qué registro está disponible.
- ▶ Se utiliza una tabla de ocupación de registros:

R1	R2	R3	R4
L	L	L	L

- ▶ Luego de generar código para el terceto, el registro donde quedó el resultado de la operación, quedará marcado como ocupado.
- ▶ Se agrega al terceto, la información del registro donde quedó el resultado.

# Tercetos → Assembler

## Seguimiento de Registros

---

- ▶ Se genera código para cada terceto, ocupando/liberando registros según corresponda.
- ▶ Se agrega al terceto, la información del registro donde quedó el resultado.





# Tercetos → Assembler

## Seguimiento de Registros

---

10. ( \* , b , c ) RI

11. ( + , a , [10] )

12. ( + , e , f )

13. ( / , d , [12] )

14. ( - , [11] , [13] )

15. ( + , [14] , 20 )

16. ( := , z , [15] )

MOV RI, \_b

MUL RI, \_c

RI	R2	R3	R4
⊥ O	L	L	L



# Tercetos $\rightarrow$ Assembler

## Seguimiento de Registros

---

10. ( \* , b , c ) R1

11. ( + , a , [10] ) R1

12. ( + , e , f )

13. ( / , d , [12] )

14. ( - , [11] , [13] )

15. ( + , [14] , 20 )

16. ( := , z , [15] )

ADD R1, \_a

R1	R2	R3	R4
Ł	L	L	L
O			



# Tercetos $\rightarrow$ Assembler

## Seguimiento de Registros

---

- 10. ( \* , b , c ) R1
- 11. ( + , a , [10] ) R1
- 12. ( + , e , f ) R2
- 13. ( / , d , [12] )
- 14. ( - , [11] , [13] )
- 15. ( + , [14] , 20 )
- 16. ( := , z , [15] )

R1	R2	R3	R4
⊥	⊥	L	L
0	0		

```
MOV R2, _e  
ADD R2, _f
```



# Tercetos → Assembler

## Seguimiento de Registros

---

- 10. ( \* , b , c ) R1
- 11. ( + , a , [10] ) R1
- 12. ( + , e , f ) R2
- 13. ( / , d , [12] ) R3
- 14. ( - , [11] , [13] )
- 15. ( + , [14] , 20 )
- 16. ( := , z , [15] )

R1	R2	R3	R4
⊥ ○	L	⊥ ○	L

```
MOV R3, _d  
DIV R3, R2
```



# Tercetos $\rightarrow$ Assembler

## Seguimiento de Registros

---

- 10. ( \* , b , c ) R1
- 11. ( + , a , [10] ) R1
- 12. ( + , e , f ) R2
- 13. ( / , d , [12] ) R3
- 14. ( - , [11] , [13] ) R1
- 15. ( + , [14] , 20 )
- 16. ( := , z , [15] )

R1	R2	R3	R4
0	L	L	L

SUB R1, R3



# Tercetos $\rightarrow$ Assembler

## Variables Auxiliares

---

- 10. ( \* , b , c ) R1
- 11. ( + , a , [10] ) R1
- 12. ( + , e , f ) R2
- 13. ( / , d , [12] ) R3
- 14. ( - , [11] , [13] ) R1
- 15. ( + , [14] , 20 ) R1
- 16. ( := , z , [15] )

R1	R2	R3	R4
L O	L	L	L

ADD R1, 20



# Tercetos $\rightarrow$ Assembler

## Variables Auxiliares

---

- 10. ( \* , b , c ) R1
- 11. ( + , a , [10] ) R1
- 12. ( + , e , f ) R2
- 13. ( / , d , [12] ) R3
- 14. ( - , [11] , [13] ) R1
- 15. ( + , [14] , 20 ) R1
- 16. ( := , z , [15] )

R1	R2	R3	R4
L	L	L	L

MOV \_z, R1



# Tercetos → Assembler

## Seguimiento de Registros

---

```
MOV R1, _b
MUL R1, _c
ADD R1, _a
MOV R2, _e
ADD R2, _f
MOV R3, _d
DIV R3, R2
SUB R1, R3
ADD R1, 20
MOV _z, R1
```

10 instrucciones  
Y  
ninguna variable  
auxiliar



Código más chico



Código más rápido





# Tercetos → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 1:

Operación entre 2 variables y/o constantes

( OP , vble1 , vble2 )

### Situación 2:

Operación entre un registro y una variable o constante

( OP , RI , vble )

### Situación 3:

Operación entre dos registros

( OP , reg1 , reg2 )

### Situación 4.a:

Operación conmutativa entre una variable (o constante) y un registro

( \* , vble , reg )

### Situación 4.b:

Operación no conmutativa entre una variable (o constante) y un registro

( - , vble , reg )



Generación de Código



# Tercetos → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 1:

Operación entre 2 variables y/o  
constantes

( OP , vble1 , vble2 )

1. Ocupar un registro
2. Generar código sobre ese registro

R1	R2	R3	R4
L	L	L	L

MOV R1, vble1

OP R1, vble2



# Tercetos → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 2:

Operación entre un registro y una variable o constante

( OP , RI , vble )

1. Generar código sobre ese registro  
(Tabla sin cambios)

OP RI, vble

RI	R2	R3	R4
O	L	L	L



# Tercetos → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 3:

Operación entre dos registros

( OP , reg1 , reg2 )

1. Generar código sobre el primer registro
2. Liberar el segundo registro

OP R1, R2

R1	R2	R3	R4
O	⊖ L	L	L



# Tercetos → Assembler

## Seguimiento de Registros

### Decisiones del Algoritmo:

---

#### Situación 4:

Operación entre una variable (o constante) y un registro

( OP , vble , reg )



# Tercetos → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 4.a:

Operación conmutativa entre una variable (o constante) y un registro

( \* , vble , reg )

1. Generar código sobre el registro

(Tabla sin cambios)

MUL R1, vble

R1	R2	R3	R4
O	L	L	L



Generación de Código

# Tercetos → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Situación 4.b:

Operación no conmutativa entre una variable (o constante) y un registro

( - , vble , reg )

1. Ocupar nuevo registro
2. Generar código sobre el nuevo registro
3. Liberar el primer registro

RI	R2	R3	R4
○	⊥ ○	L	L

MOV R2, vble

SUB R2, RI

RI	R2	R3	R4
⊖ L	○	L	L



# Tercetos → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Asignaciones – Situación a:

Valor a asignar en un registro

( := , vble , reg )

1. Generar código usando el registro
2. Liberar el registro

MOV vble, R1

R1	R2	R3	R4
⊖ L	L	L	L



Generación de Código



# Tercetos → Assembler

## Seguimiento de Registros

Decisiones del Algoritmo:

---

### Asignaciones – Situación b:

Valor a asignar en una variable (o constante)

( := , vble1 , vble2 )

1. Ocupar registro
2. Generar código usando el registro
3. Liberar el registro

R1	R2	R3	R4
⊥	L	L	L
0			

MOV R1, vble2

MOV vble1, R1

R1	R2	R3	R4
⊖	L	L	L
L			

---

Generación de Código

# Seguimiento de Registros

---

- ▶ El **número máximo de registros** necesarios depende del número de niveles de precedencia:

- ▶ Con 2 niveles ( + y -, \* y / ) → 2 registros
- ▶ Con 3 niveles ( potencia ) → 3 registros
- ▶ Con 4 niveles ( < y > ) → 4 registros
- ▶ Con 5 niveles ( AND y OR ) → 5 registros



# Seguimiento de Registros

---

- ▶ Si se necesita un registro y no hay más, se libera un registro, moviendo su contenido a una variable auxiliar.
- ▶ Se posterga el uso de registros que tienen usos específicos (sirven para más operaciones).
- ▶ Ejemplo: En Pentium, EAX, EBX, ECX, EDX
- ▶ EAX y EDX se utilizan para multiplicaciones y divisiones  
→ Comenzar utilizando los otros 2.



¿PREGUNTAS?