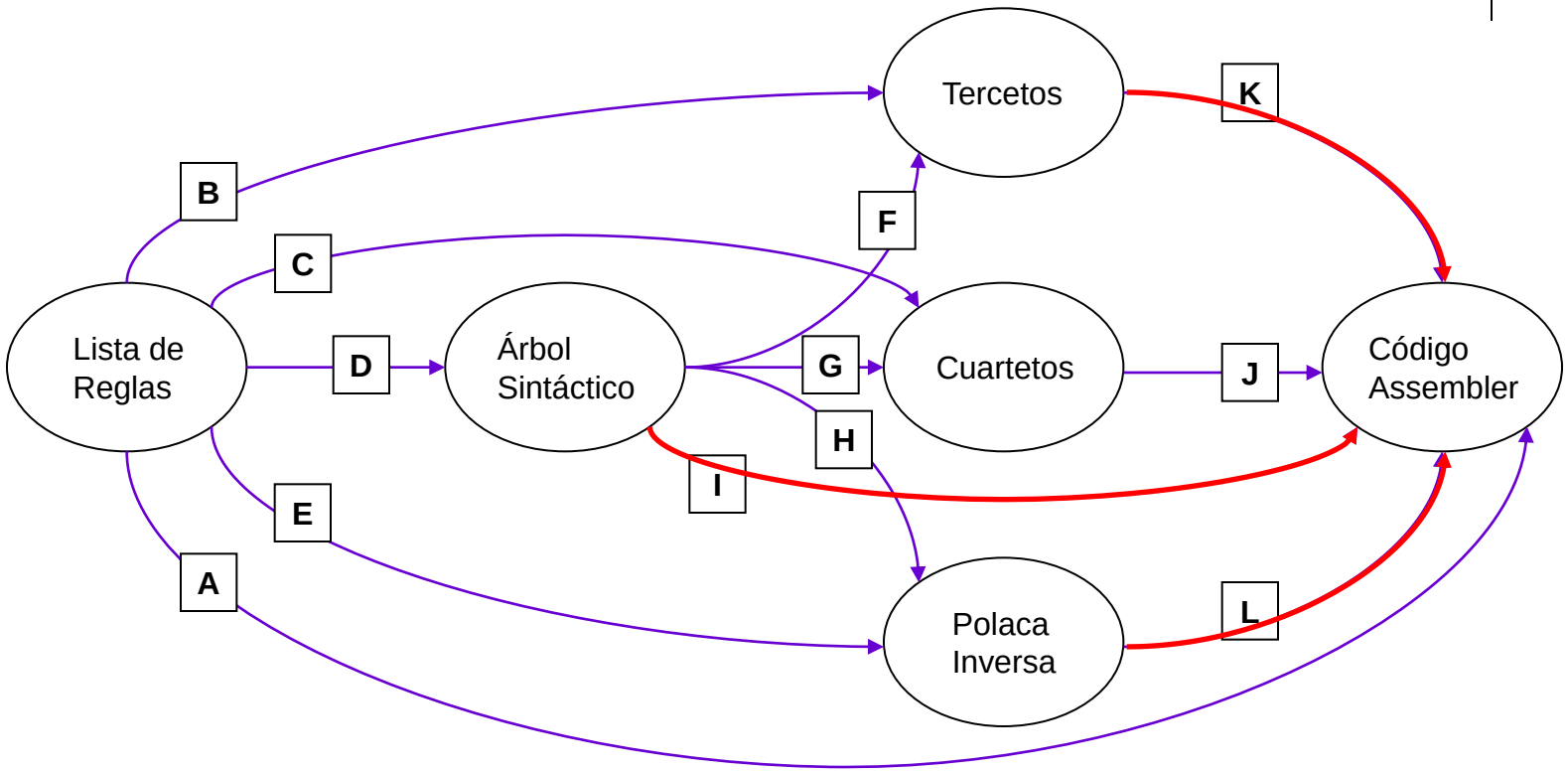
Lenguajes y Compiladores

Generación de código assembler

UNLaM

Generación de Código



Caminos posibles:

Camino 1: A
Camino 2: D, I
Camino 3: E, L
Camino 4: C, J
Camino 5: B, K
Camino 6: D, F, K
Camino 7: D, G, J
Camino 8: D, H, L

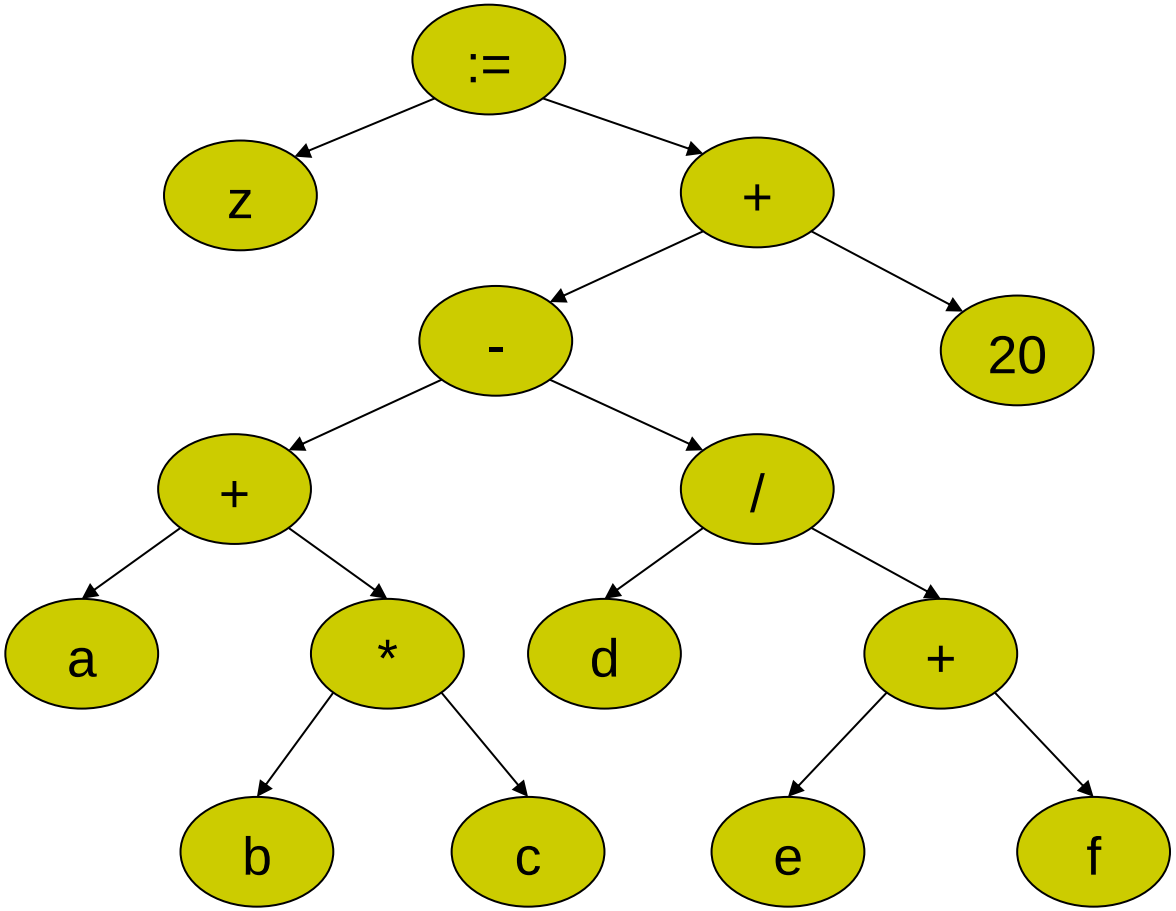
Traducciones

- Árbol Sintáctico → Assembler
- Tercetos → Assembler
- Polaca Inversa → Assembler

ÁRBOL SINTÁCTICO→ ASSEMBLER

Árbol Sintáctico → Assembler

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

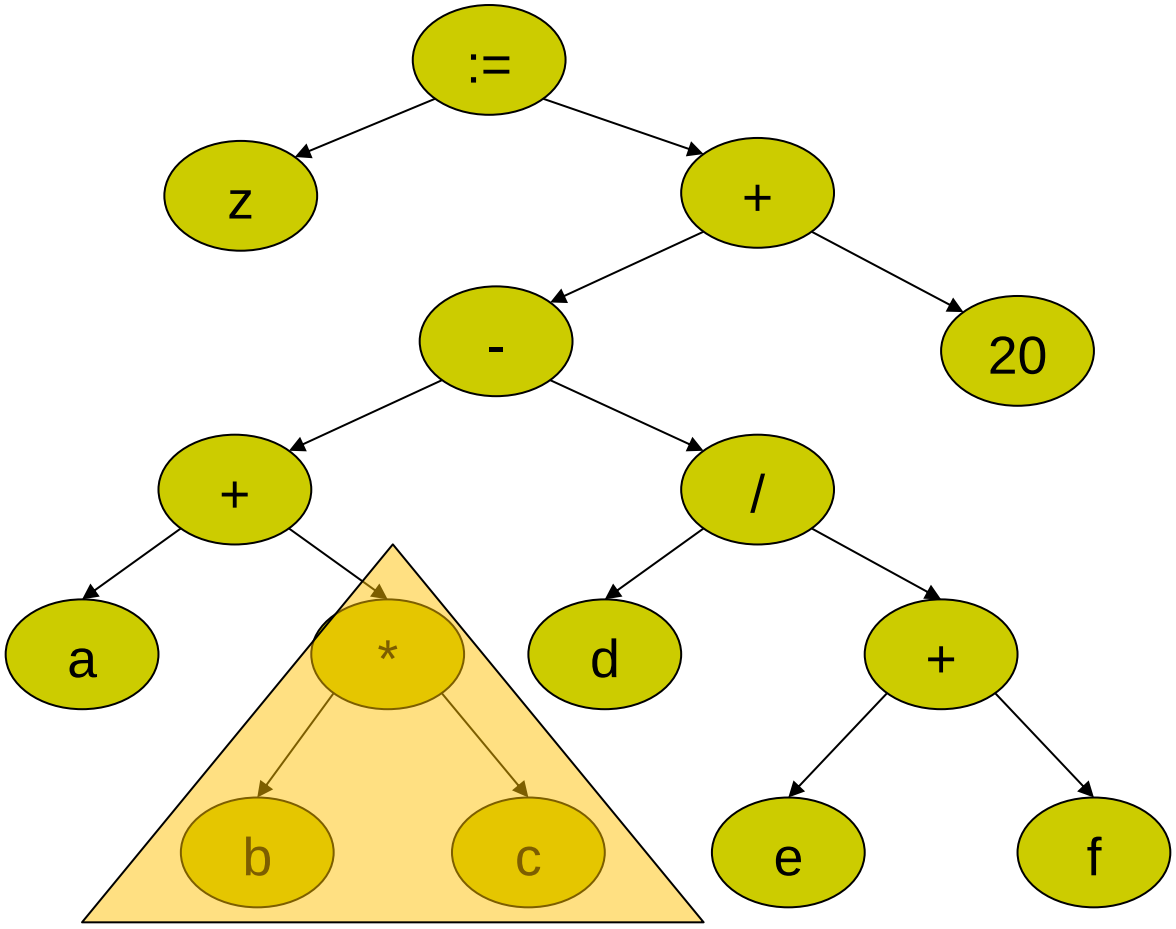


Árbol Sintáctico → Assembler

- 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




```
MOV R1, _b
MUL R1, _c
MOV @aux1, R1
```

Árbol Sintáctico → Assembler


```
MOV R1, _b
MUL R1, _c
MOV @aux1, R1
```

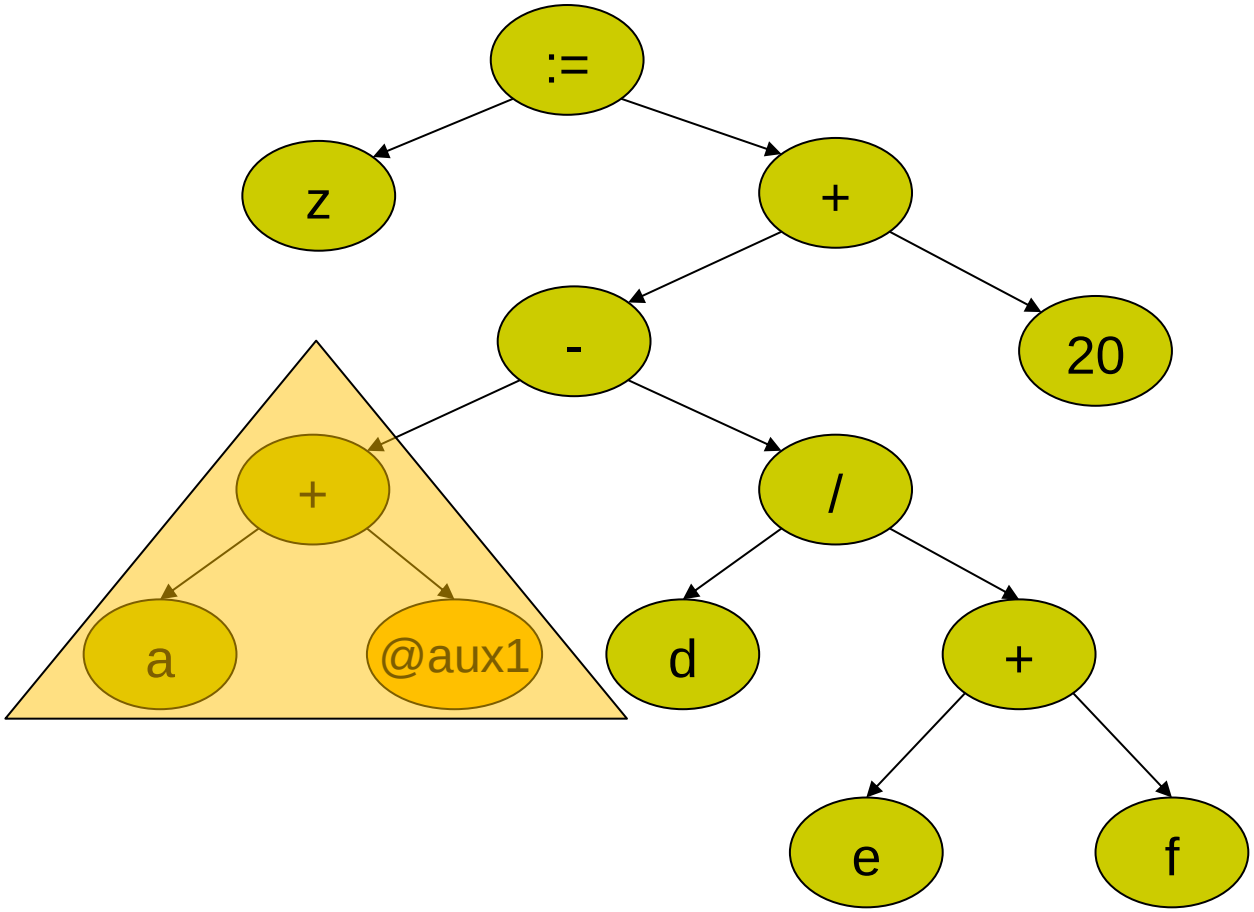
- El texto marcado en rojo es texto fijo.
- Se agregan sufijos (_, @) a las variables para evitar errores en la compilación del código Assembler.

  Por ejemplo:

  x := MUL * k

  MOV R1, MUL ← Error de compilación
- Los sufijos 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

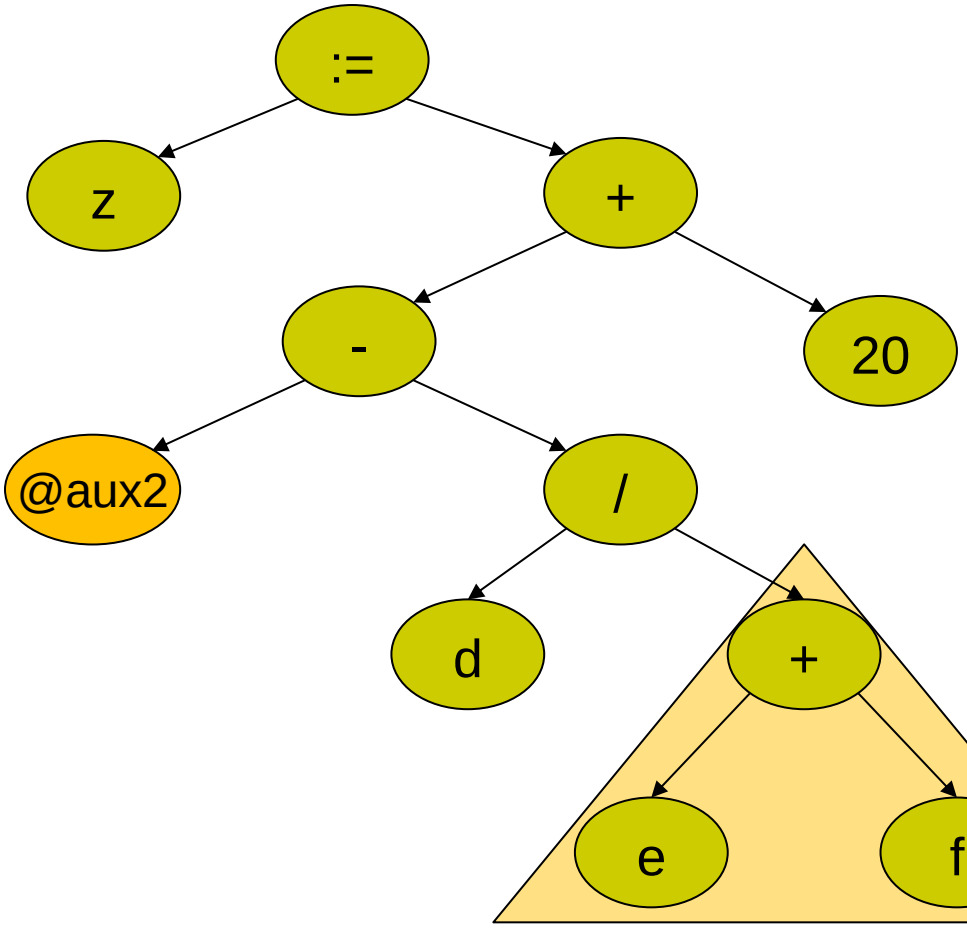


MOV R1, _a

ADD R1, @aux1

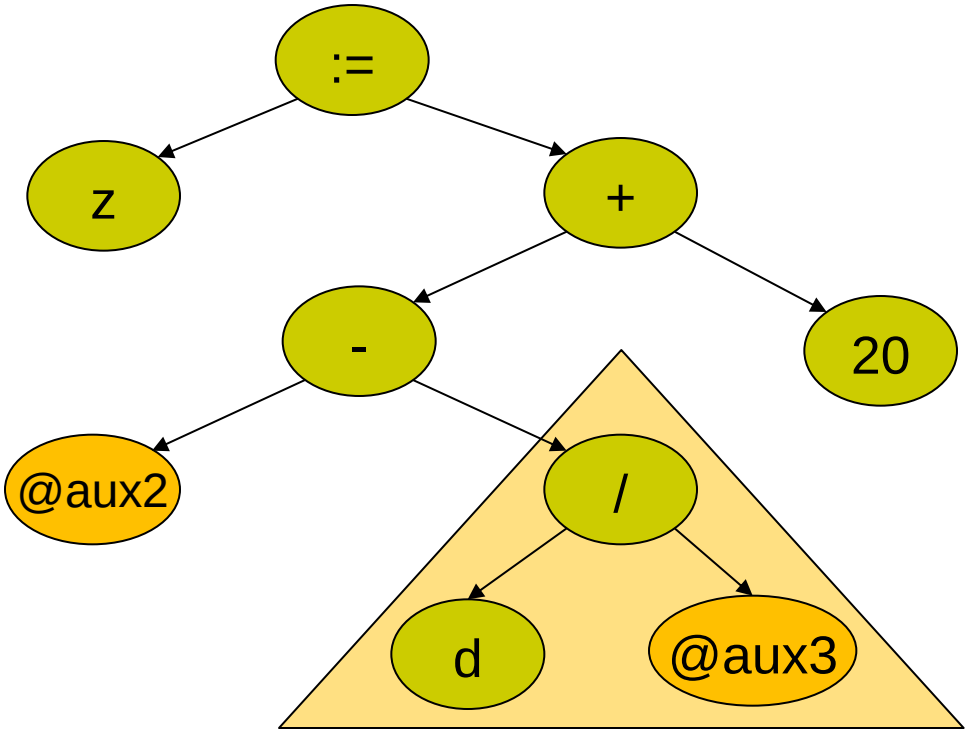
MOV @aux2, R1

Árbol Sintáctico → Assembler




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

Árbol Sintáctico → Assembler

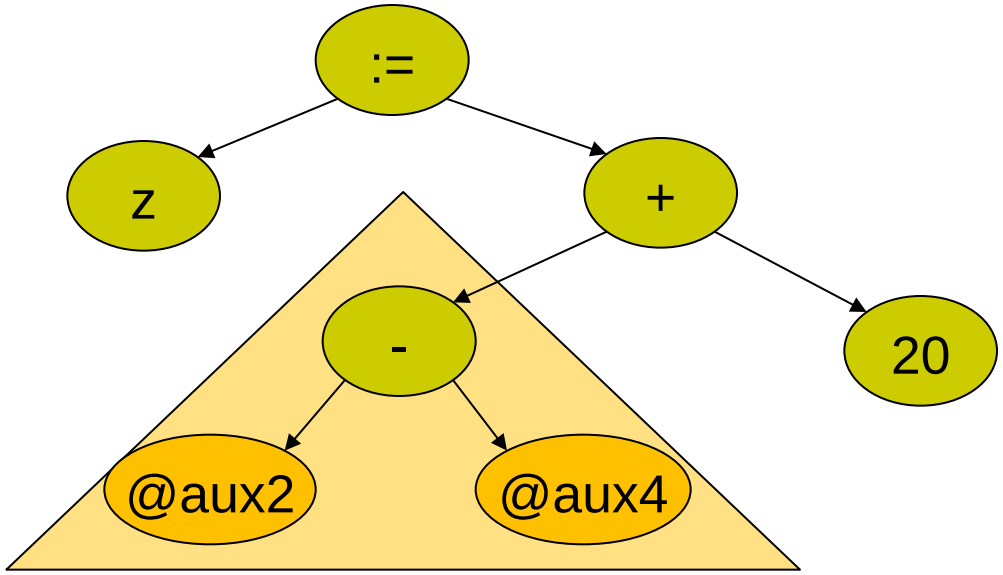


MOV R1, _d

DIV R1, @aux3

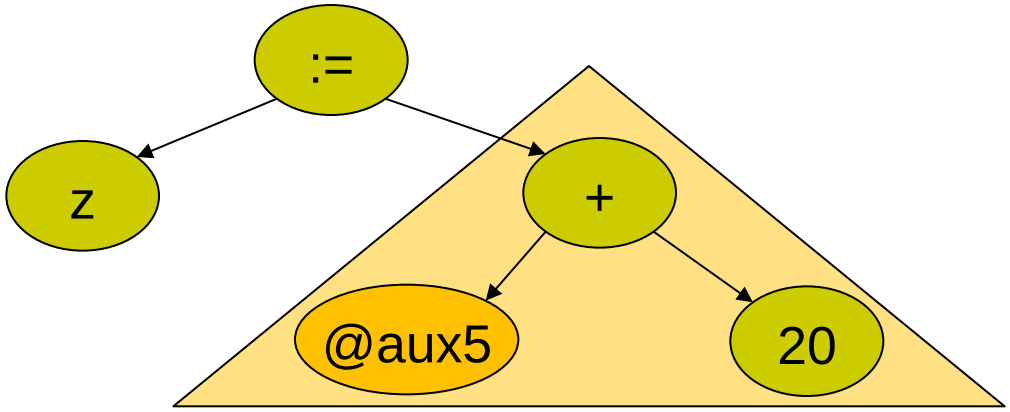
MOV @aux4, R1

Árbol Sintáctico → Assembler




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

Árbol Sintáctico → Assembler

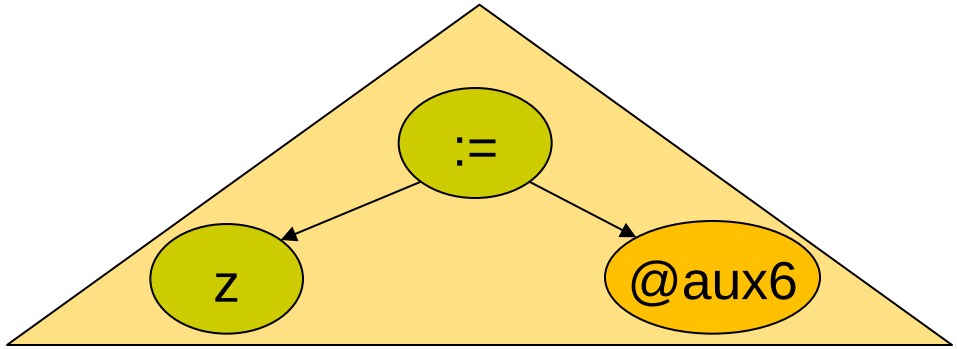


MOV R1, @aux5

ADD R1, 20

MOV @aux6, R1

Árbol Sintáctico → Assembler



MOV R1, @aux6

MOV _z, R1

Árbol Sintáctico → Assembler


```
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
```

TERCETOS→ ASSEMBLER

Tercetos → 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

- 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

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

```
MOV R1, _b
MUL R1, _c
MOV @aux1, R1
```

Tercetos → Assembler

MOV R1, _b
MUL R1, _c
MOV @aux1, R1

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

  Por ejemplo:

  x := MUL * k

  MOV R1, MUL ← Error de compilación
- Los sufijos para las variables auxiliares deben ser diferentes a los de las variables del usuario.
- Las variables auxiliares se guardan en la TdeS.

Tercetos → Assembler

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

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

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 R1, _d
DIV R1, @aux3
MOV @aux4, R1
```

Tercetos → Assembler

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

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

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
```

Tercetos → Assembler


```
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
```

POLACA INVERSA → ASSEMBLER

Polaca Inversa → Assembler

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

Árbol Sintáctico → Polaca Inversa

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

Lista de Reglas → Polaca Inversa

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

Polaca Inversa → Assembler

- 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

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

Leído	Pila
z	z
a	z a
b	z a b
c	z a b c
*	z a
*****	**z a @aux1**

MOV R1, _b

MUL R1, _c

MOV @aux1, R1

Polaca Inversa → Assembler


```
MOV R1, _b
MUL R1, _c
MOV @aux1, R1
```

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

  Por ejemplo:

  x := MUL * k

  MOV R1, MUL ← Error de compilación
- Los sufijos 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

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

Leído	Pila
	z a @aux1
+	z
+	**z @aux2**

```
MOV R1, _a
ADD R1, @aux1
MOV @aux2, R1
```

Polaca Inversa → Assembler

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

Leído	Pila
	z @aux2
d	z @aux2 d
e	z @aux2 d e
f	z @aux2 d e f
+	z @aux2 d
+	**z @aux2 d @aux3**

MOV R1, _e

ADD R1, _f

MOV @aux3, R1

Polaca Inversa → Assembler

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

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

```
MOV R1, _d
DIV R1, @aux3
MOV @aux4, R1
```

Polaca Inversa → Assembler

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

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

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

Leído	Pila
	z @aux6
:=	-
:=	**-**

MOV R1, @aux6

MOV _z, R1

Polaca Inversa → Assembler


```
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
```

Generación de Código

¿Preguntas?