

# TALLER DE DISEÑO DE SOFTWARE (3306)

## Código Intermedio

Francisco Bavera

pancho@dc.exa.unrc.edu.ar

<http://dc.exa.unrc.edu.ar/nuevodc/materias/compiladores>



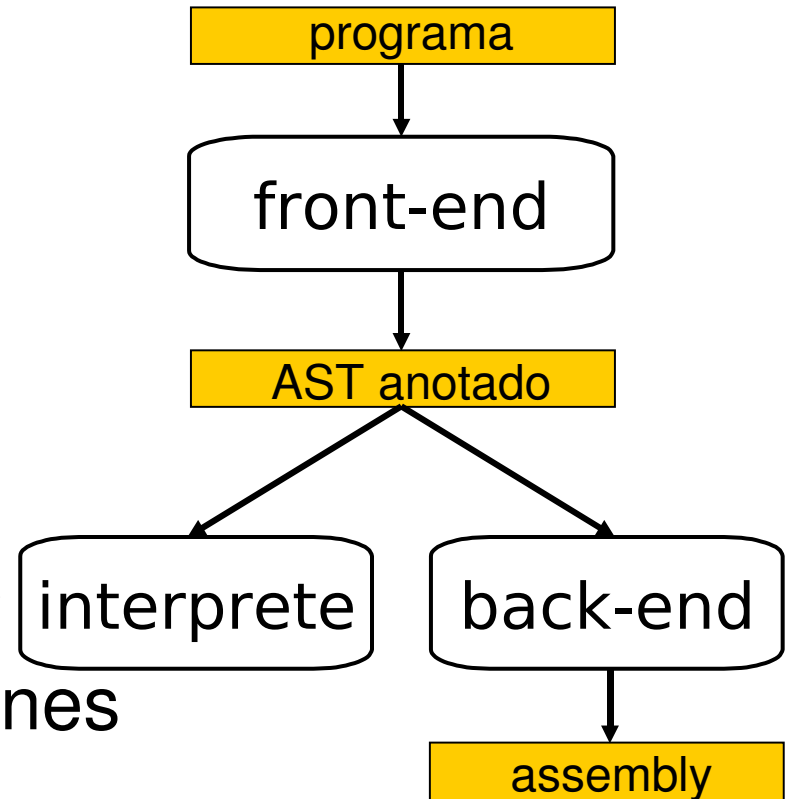
**Grupo Procesadores de Lenguajes**  
**Departamento de Computación**  
**Facultad de Ciencias Exactas, Físico-Químicas y Naturales**  
**Universidad Nacional de Río Cuarto**

# Overview

- Interpretación

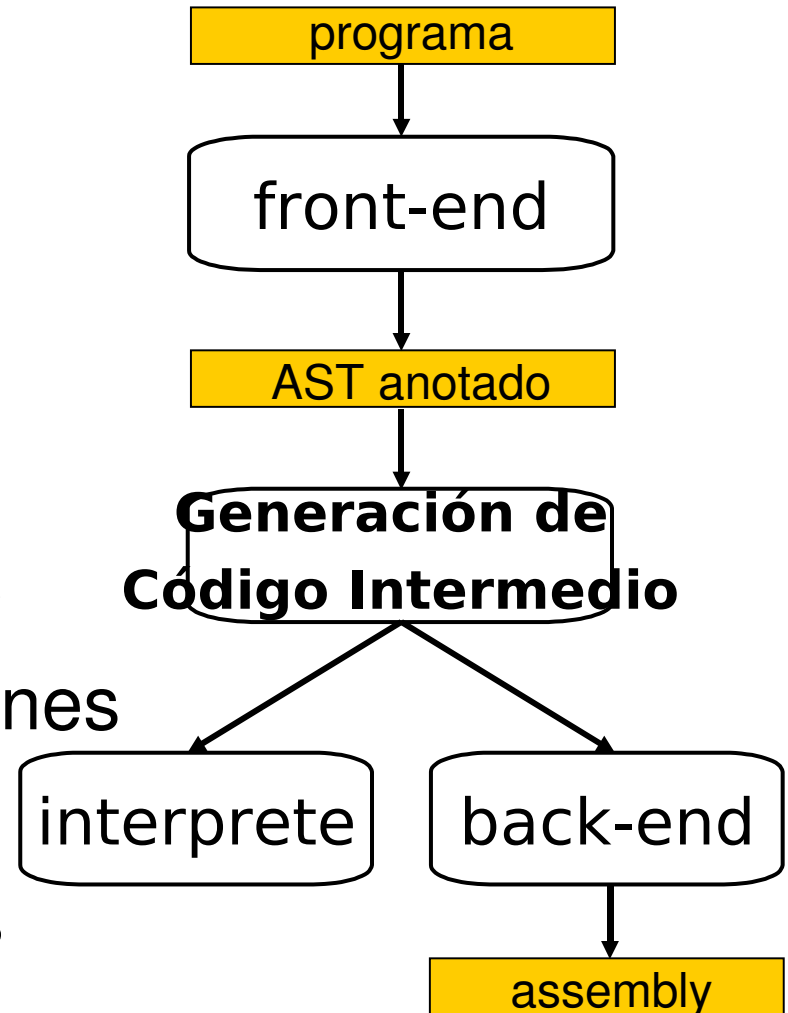
- Generación de Código

- Selección de instrucciones
- Elección d e registros
- Orden de instrucciones



# Overview

- Código Intermedio
- Interpretación
- Generación de Código
  - Selección de instrucciones
  - Elección d e registros
  - Orden de instrucciones



# Interpretación

- interpretación **recursiva**
  - Opera directamente sobre el AST [attribute grammar]
  - simple de escribir
  - Poca verificación de errores
  - Poco eficiente
- interpretación **iterativa**
  - Opera sobre código intermedio
  - Mejor verificación de errores
  - Mejor eficiencia



# Generación de código intermedio

- La representación intermedia se puede considerar como un programa para una máquina abstracta
- La representación debe ser fácil de producir
- La representación debe ser fácil de traducir a objeto (el código máquina de un procesador concreto)

# Generación de código intermedio

- Independiente del lenguaje
  - Sin tipos estructurados , solo tipos básicos (char, int, float)
  - Sin estructuras de control de flujo, solo saltos (in)condicionales
- Formato lineal
  - Java bytecode

# Generación de código intermedio

- Arbol de parsing
- Arbol sintáctico Abstracto
- Código de tres direcciones
- Maquina Pila

# Generación de Código Intermedio

## Reescritura del árbol

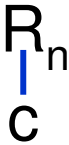
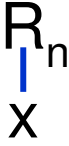
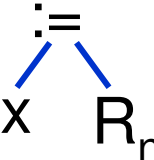



- Remplazar nodos y subárboles del AST por segmentos de código intermedio
- Producir una secuencia lineal de instrucciones a partir del árbol reescrito

## Ejemplo

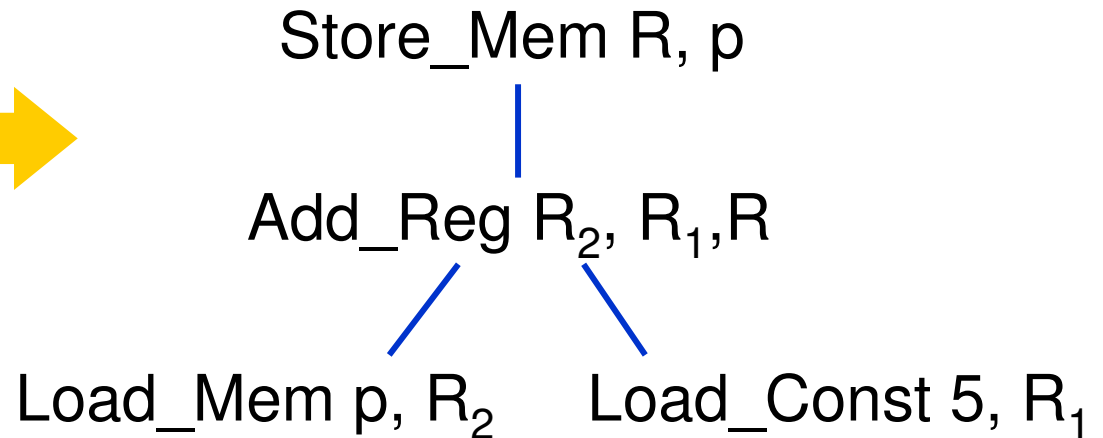
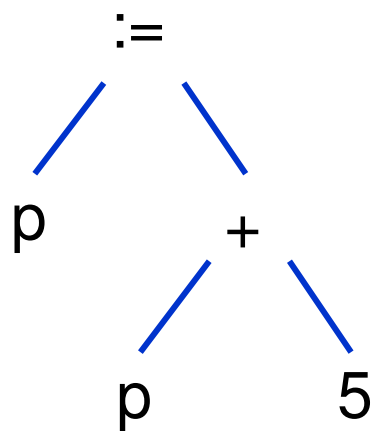
- código:  $p := p + 5$
- destino: código de tres direcciones



# Instrucciones código de tres direcciones

Instrucción	Acción	pattern
Load_Const $c, R_n$	$R_n := c$	
Load_Mem $x, R_n$	$R_n := x$	
Store_Mem $R_n, x$	$x := R_n$	
Add_Reg $R_m, R_n, R$	$R := R_n + R_m$	
Sub_Reg $R_m, R_n, R$	$R := R_n - R_m$	
Mul_Reg $R_m, R_n, R$	$R := R_n * R_m$	

# Reescritura del árbol para $p := p + 5$



- Secuencia lineal:

Load\_Mem  $p, R_2$   
Load\_Const  $5, R_1$   
Add\_Reg  $R_2, R_1$   
Store\_Mem  $R_1, p$

# Generación de Código Intermedio

## Reescritura del árbol

- Remplazar nodos y subárboles del AST por segmentos de código intermedio
- Producir una secuencia lineal de instrucciones a partir del árbol reescrito

## Ejemplo

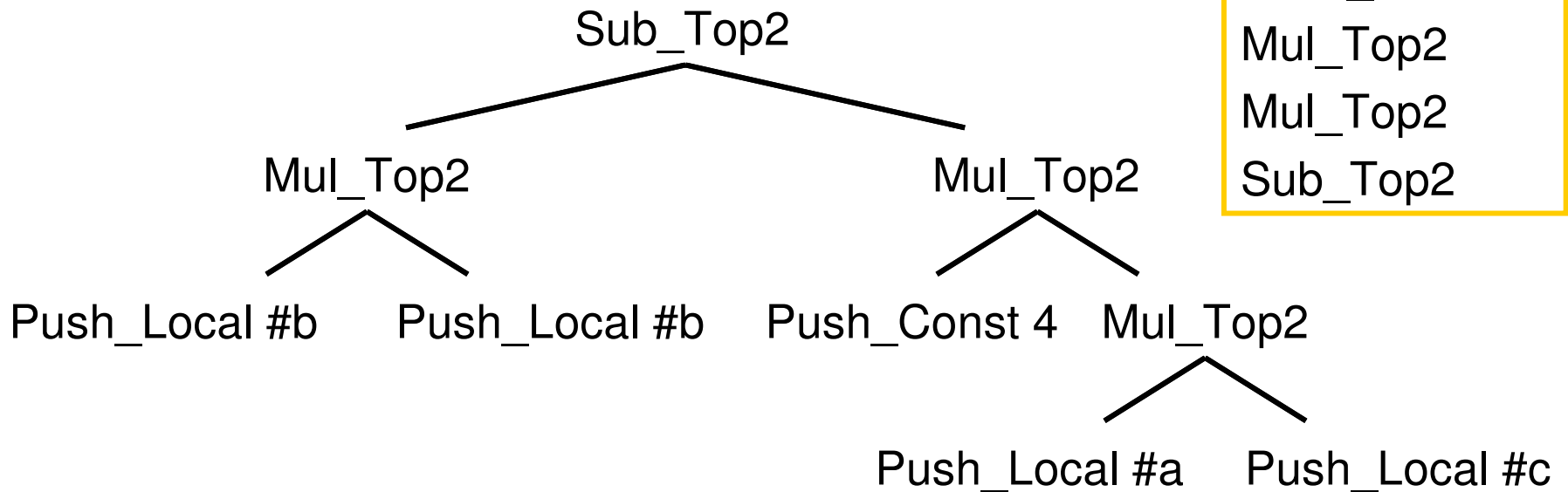
- código:  $b*b - 4*a*c$
- destino: máquina pila

# Instrucciones máquina pila

Instrucción	Acción
Push_Const    c	SP++; stack[SP] = c;
Push_Local    i	SP++; stack[SP] = i;
Store_Local    i	i = stack[SP]; SP--;
Add_Top2	stack[SP-1] = stack[SP-1] + stack[SP]; SP--;
Sub_Top2	stack[SP-1] = stack[SP-1] - stack[SP]; SP--;
Mul_Top2	stack[SP-1] = stack[SP-1] * stack[SP]; SP--;

# Generación de código intermedio para una máquina pila

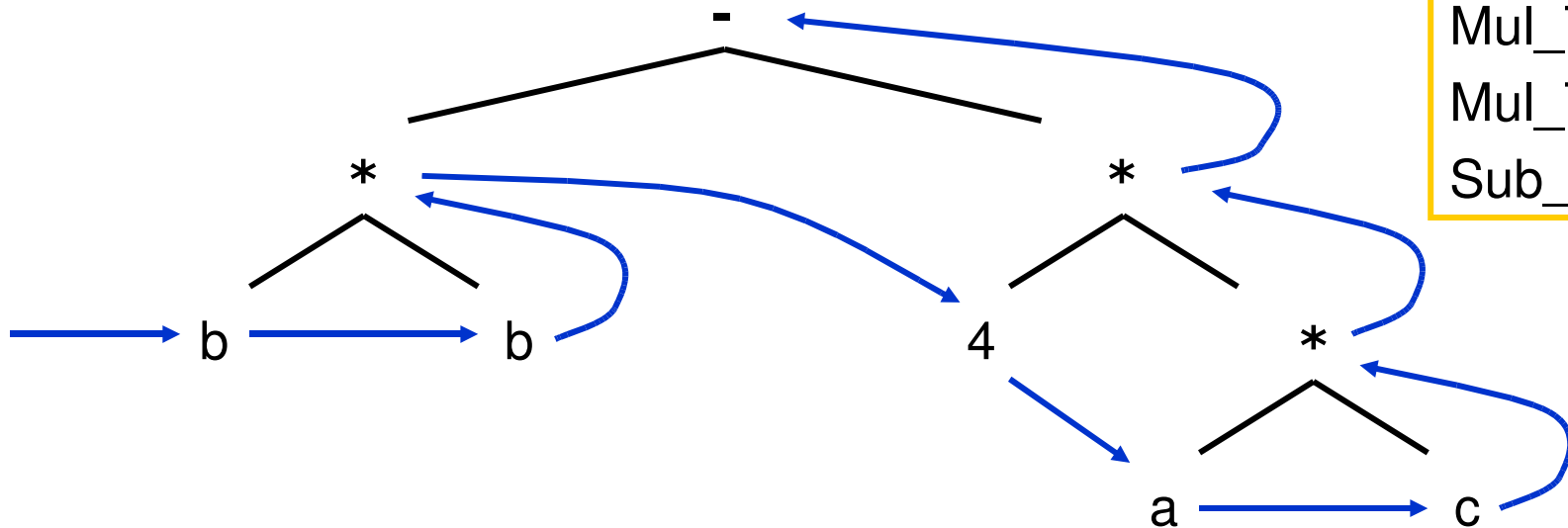
- ejemplo:  $b*b - 4*a*c$
- AST



# Generación de código intermedio para una máquina pila

- ejemplo:  $b*b - 4*a*c$
- Recorrido del AST

```
Push_Local #b  
Push_Local #b  
Mul_Top2  
Push_Const 4  
Push_Local #a  
Push_Local #c  
Mul_Top2  
Mul_Top2  
Sub_Top2
```





# Resumen

- Interpretación
  - recursiva
  - iterativa
- Generación código intermedio
  - Código tres direcciones
  - Maquina Pila

