# UNIVERSIDAD SAN PABLO DE GUATEMALA

Facultad de Ciencias Empresariales

Escuela de Ingeniería

Ingeniería en Sistemas Ciencias y de la Computación


## Tarea de Semana 16

Tarea presentada en el curso de Compiladores

Impartido por: ING. Walter Oswaldo Ordoñez Sierra

Yordy Estiven Rodríguez Morales - 2300321

Guatemala, 07 de noviembre 2025

## Introducción

En el proceso de compilación, la generación de código intermedio es una de las fases más importantes, ya que actúa como un puente entre la traducción del código fuente y la producción del código máquina final.
El compilador transforma el código escrito por el programador en una representación más abstracta, conocida como **código intermedio**, que permite aplicar optimizaciones y simplificar la generación del código final.

Gracias a este paso, los compiladores modernos logran que los programas sean más eficientes y puedan ejecutarse en distintas plataformas sin tener que volver a analizar o interpretar todo el código fuente. Además, facilita la detección de errores y la mejora del rendimiento a través de técnicas de optimización antes de generar el ejecutable final.

## Marco Teórico

#### 1. Etapas del compilador

Un compilador moderno se compone de varias fases principales:

1. **Análisis léxico:** identifica los tokens del programa (palabras clave, identificadores, símbolos, etc.).
2. **Análisis sintáctico:** organiza los tokens según las reglas gramaticales del lenguaje.
3. **Análisis semántico:** verifica que el código tenga sentido lógico (por ejemplo, tipos compatibles).
4. **Generación de código intermedio:** traduce el árbol sintáctico en una representación más simple y manipulable.
5. **Optimización:** mejora la eficiencia del código sin alterar su comportamiento.
6. **Generación de código final:** convierte la representación optimizada en código máquina.

#### 2. Representación intermedia

El **código intermedio** es una forma neutral del programa que permite al compilador trabajar de manera más flexible.
Algunos tipos comunes son:

- **Árbol de sintaxis abstracta (AST):** estructura jerárquica que representa la organización del código.
- **Código de tres direcciones (Three-Address Code, TAC):** representa operaciones simples con a lo sumo tres operandos (por ejemplo, `t1 = a + b`).
- **Representación intermedia basada en registros:** usada en compiladores como LLVM.

El código de tres direcciones es uno de los formatos más usados, ya que simplifica las expresiones complejas en pasos más pequeños, facilitando su análisis y optimización.

### 3. Control de flujo

El **control de flujo** describe el orden en que se ejecutan las instrucciones. Durante la generación del código intermedio, el compilador construye **bloques básicos** (secciones de código que siempre se ejecutan en secuencia) y un **grafo de flujo de control (CFG)** que conecta estos bloques según las condiciones del programa. Esto es clave para aplicar optimizaciones posteriores como la eliminación de saltos innecesarios o la reorganización de bucles.

### 4. Optimización de código

La optimización busca mejorar el rendimiento del programa sin alterar su lógica. Algunas técnicas comunes son:

- **Optimización algebraica:** reemplaza operaciones costosas por equivalentes más simples (por ejemplo, $x * 2 \rightarrow x + x$).
- **Eliminación de código muerto:** elimina instrucciones que no afectan el resultado final.
- **Propagación de constantes:** sustituye variables cuyo valor es conocido por su constante.
- **Optimización de bucles:** mejora la eficiencia en iteraciones, por ejemplo moviendo cálculos fuera del bucle.

Estas transformaciones permiten que el programa final consuma menos recursos, sea más rápido y eficiente.

## Conclusiones

La generación de código intermedio es una fase esencial en el proceso de compilación, ya que permite separar la lógica del lenguaje fuente del lenguaje máquina. Gracias a esta representación, el compilador puede aplicar transformaciones que mejoran la eficiencia del programa antes de producir el código final.

Además, el uso de estructuras como el código de tres direcciones facilita la depuración, la optimización y la portabilidad entre diferentes plataformas. Las técnicas de optimización —como la eliminación de código muerto o la propagación de constantes— son fundamentales para obtener programas más rápidos y con menor consumo de recursos.

## Referencias


- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, Techniques, and Tools* (2nd ed.). Pearson Education.
- Muchnick, S. S. (1997). *Advanced Compiler Design and Implementation*. Morgan Kaufmann.
- Cooper, K., & Torczon, L. (2011). *Engineering a Compiler* (2nd ed.). Morgan Kaufmann.