



INTERPRETE

CODIGO INTERMEDIO

POR:

- CESAR EDUARDO ELIAS DEL HONO
- JOSE LUIS SANDOVAL PEREZ
- DIEGO EMANUEL SAUCEDO ORTEGA
- CARLOS DANIEL TORRES MACIAS



DEFINICIÓN

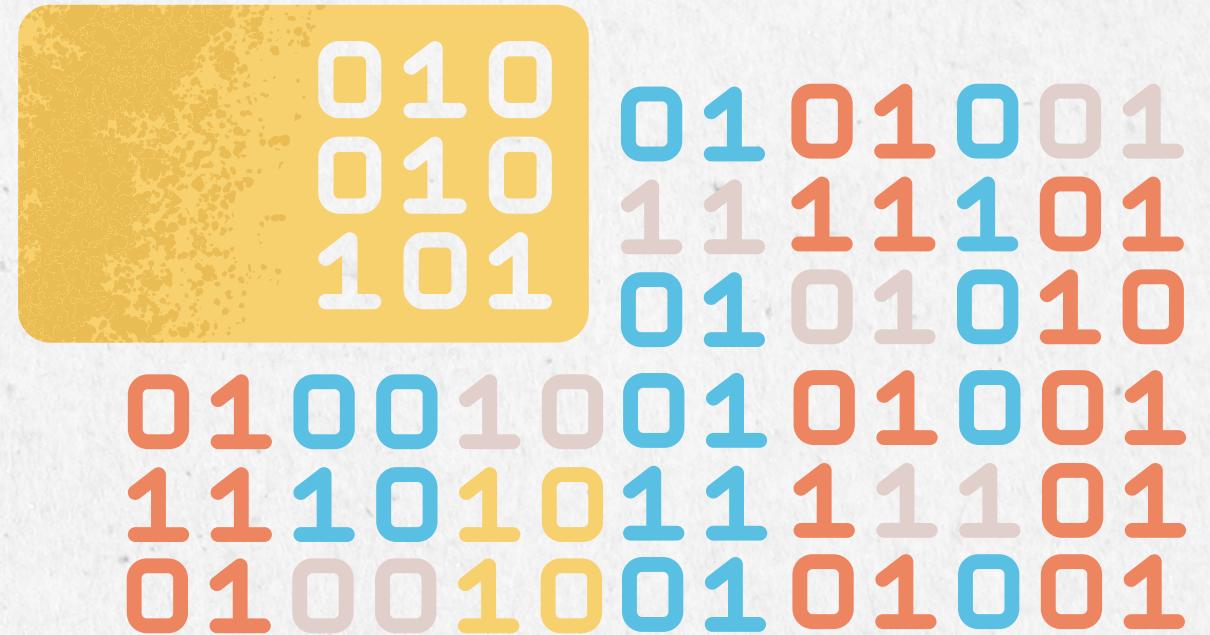


graphic designer

- Es un programa que toma instrucciones de un lenguaje intermedio, generado a partir de un lenguaje de alto nivel y las ejecuta directamente.
- Este código “intermedio” es un formato que suele ser más abstracto que el código máquina, pero más sencillo que el código fuente original.

FUNCIONES

- **Transporte entre compilación y ejecución:** Sirve de paso entre el código fuente y la máquina objetivo, facilitando la optimización y la portabilidad.
- **Optimización multiplataforma:** Permite optimizaciones antes de la conversión final a código máquina para adaptarse a distintas arquitecturas de hardware.
- **Eficiencia en compilación JIT (Just-In-Time):** Muchos lenguajes, como Java y C#, utilizan intérpretes para convertir el código intermedio en código máquina en tiempo de ejecución, lo que permite ejecutar el programa sin una compilación previa completa.



CARACTERÍSTICAS

- **Portabilidad:**

Permiten ejecutar el mismo código en distintas plataformas sin modificaciones.

- **Optimización en Tiempo de Ejecución:**

Facilitan la optimización dinámica durante la ejecución, mejorando el rendimiento.

- **Facilidad de Depuración:**

Ejecutan el código instrucción por instrucción, lo que facilita la identificación de errores en tiempo real.

- **Eficiencia en Recursos:**

No requieren la compilación completa del programa, lo que reduce el uso de recursos antes de la ejecución.



COMPONENTES

1. Contexto de Ejecución:

Simula el entorno donde se ejecuta el programa, incluyendo la memoria, la pila de ejecución y la tabla de símbolos.

2. Memoria:

Almacena variables y valores del programa. Se estructura como una lista indexada por direcciones y contiene tanto valores como direcciones.

3. Pila de Ejecución:

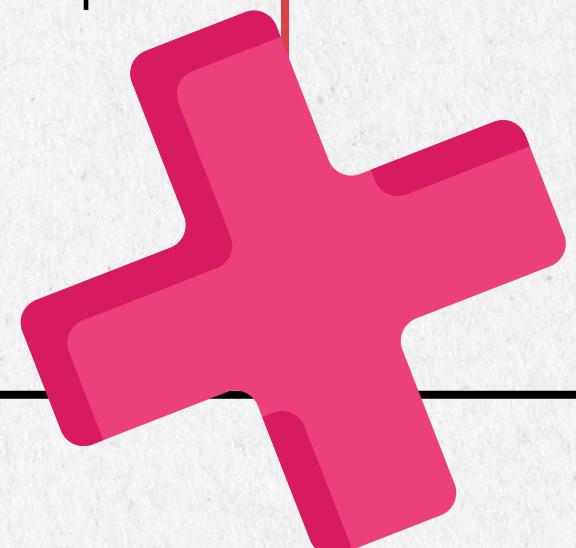
Estructura tipo LIFO donde se almacenan temporalmente los valores intermedios. Las instrucciones operan directamente sobre esta pila para realizar cálculos.

4. Tabla de Símbolos:

Contiene la información sobre variables y etiquetas. Permite acceder rápidamente a las direcciones de memoria durante la ejecución.

5. Registros:

Incluyen el contador de instrucciones (para seguir el flujo del programa) y el puntero de pila (que señala la posición en la pila). Ayudan a controlar la ejecución del programa.

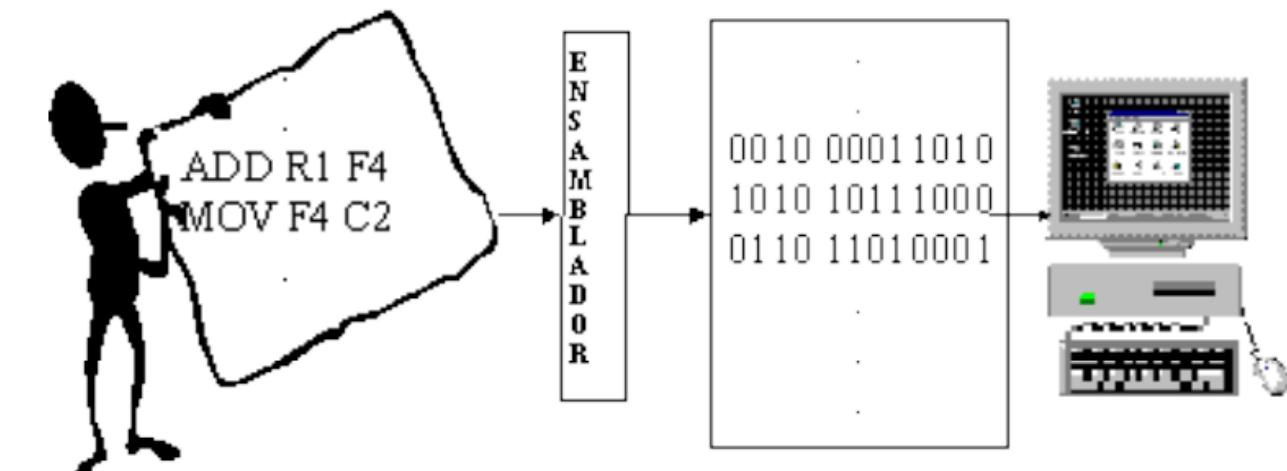


IMPLEMENTACIÓN

- Cada instrucción en el lenguaje intermedio es una subclase de una clase abstracta llamada Code.
- ****Método exec():**** Implementa la lógica de ejecución de cada instrucción.
- El programa es una secuencia de instrucciones, que se ejecutan secuencialmente a menos que haya saltos controlados.

#NOTEVANASGAGO
CÓDIGO INTERMEDIO

Se genera código ensamblador



ANLISIS LÉXICO Y SINTÁCTICO

- El código fuente debe ser analizado para obtener el código intermedio.
- *Análisis Léxico:* Identifica tokens o símbolos significativos.
- *Análisis Sintáctico:* Construye el árbol sintáctico del programa y valida su estructura gramatical.

A

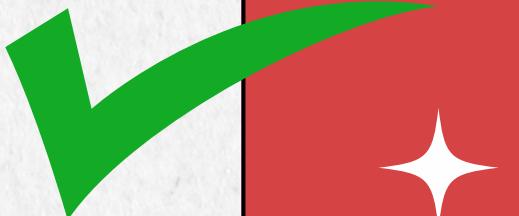
B

?

C

VENTAJAS

- 1. Flexibilidad: Permite la ejecución en múltiples plataformas sin modificar el código fuente.
- 2. Optimización en tiempo de ejecución: Se pueden aplicar optimizaciones dinámicas en la ejecución.
- 3. Facilita la depuración: Al estar a medio camino entre el código fuente y el código máquina, es más fácil identificar y corregir errores.





DESVENTAJAS



- 1. Rendimiento: Generalmente más lentos que los compiladores que generan código máquina directamente.
- 2. Consumo de memoria: Necesita estructuras adicionales como la tabla de símbolos y la pila de ejecución.
- 3. Complejidad: El diseño de un buen intérprete requiere un análisis detallado del flujo de ejecución.



EJEMPLOS DE USO

- Java (JVM): El código fuente de Java se compila en bytecode, un tipo de código intermedio que luego es interpretado por la JVM.
- .NET (CIL): Microsoft utiliza el lenguaje intermedio común (CIL) en la plataforma .NET



EN COMPARACIÓN

INTERPRETE:

- Se ejecuta linea por linea
- No requiere compilarse, lo que lo vuelve más rápida de iniciar
- Tiene una portabilidad alta, solo requiere de contar con el interprete adecuado
- Depuración sencilla: los errores se muestran en el momento

COMPILADOR

- Se traduce el programa completo
- Requiere compilarse, por lo que tarda un tiempo para arrancar
- Menor portabilidad, esta sujeto a requerimientos
- Mayor complejidad para detectar errores específicos

CONCLUSIÓN

- Los intérpretes de código intermedio son cruciales en la construcción de sistemas de ejecución multiplataforma.
- Aunque son más lentos que los compiladores, ofrecen ventajas en términos de flexibilidad y portabilidad.



creative portfolio

