Compilador, Ensamblador y Linker

David Alejandro González Márquez

Clase disponible en: https://github.com/fokerman/computingSystemsCourse

Lenguajes

Compilados

El código se transforma a código nativo de la plataforma. Para esto debe ser compilado, ensamblado y linkeado. El resultado es un **archivo binario ejecutable** que el sistema operativo entiende cómo cargar para poder ejecutar.

Ejemplos: C, C++, C#, Go

Lenguajes

Compilados

El código se transforma a código nativo de la plataforma. Para esto debe ser compilado, ensamblado y linkeado. El resultado es un **archivo binario ejecutable** que el sistema operativo entiende cómo cargar para poder ejecutar.

Ejemplos: C, C++, C#, Go

Interpretados

El código es **leído e interpretado** como instrucciones o comandos. El programa que realiza esta acción es el intérprete.

Ejemplos: Python, Ruby, JavaScript, PHP

Lenguajes

Compilados

El código se transforma a código nativo de la plataforma. Para esto debe ser compilado, ensamblado y linkeado. El resultado es un **archivo binario ejecutable** que el sistema operativo entiende cómo cargar para poder ejecutar.

Ejemplos: C, C++, C#, Go

Interpretados

El código es **leído e interpretado** como instrucciones o comandos. El programa que realiza esta acción es el intérprete.

Ejemplos: Python, Ruby, JavaScript, PHP

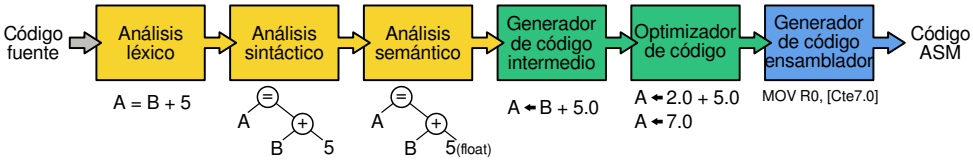
Código intermedio

El código es transformado a bytecode y ejecutado por una **máquina virtual**. Este código puede ser utilizado en múltiples plataformas. Cuentan con JIT (Just-In-Time Compiler) que consiste en transformar el bytecode a código de máquina nativo.

Ejemplos: Java, Scala, Groovy

Compilador

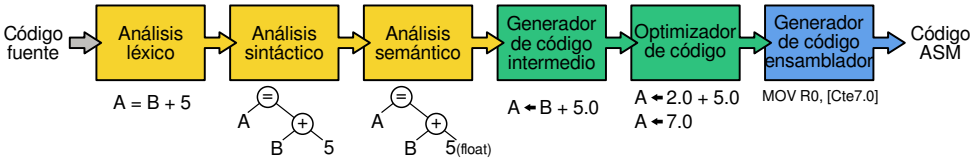
El **compilador** es un programa que traduce código en un lenguaje de programación a código en otro lenguaje de programación, generalmente lenguaje ensamblador.



El código pasa por diferentes etapas hasta generar instrucciones de ensamblador.

Compilador

El **compilador** es un programa que traduce código en un lenguaje de programación a código en otro lenguaje de programación, generalmente lenguaje ensamblador.

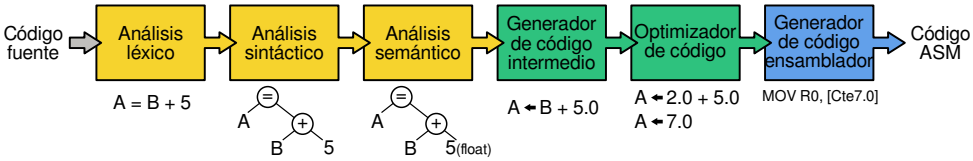


El código pasa por diferentes etapas hasta generar instrucciones de ensamblador.

En algunos lenguajes, existen procesos anteriores al compilador,
como el **precompilador** o la resolución de **macros**.

Compilador

El **compilador** es un programa que traduce código en un lenguaje de programación a código en otro lenguaje de programación, generalmente lenguaje ensamblador.



El código pasa por diferentes etapas hasta generar instrucciones de ensamblador.

En algunos lenguajes, existen procesos anteriores al compilador,
como el **precompilador** o la resolución de **macros**.

En la etapa de **optimización** el código es transformado para realizar la misma acción pero de forma diferente a la escrita por el programador, potencialmente obteniendo mejor rendimiento.

Ensamblador

El **ensamblador** transforma instrucciones ASM en código binario (código objeto).

Ensamblador

El **ensamblador** transforma instrucciones ASM en código binario (código objeto).

Esta transformación es directa, no requiere interpretar qué acciones realiza el programa.

Ensamblador

El **ensamblador** transforma instrucciones ASM en código binario (código objeto).

Esta transformación es directa, no requiere interpretar qué acciones realiza el programa.

Cuenta con **pseudoinstrucciones** que le indican al ensamblador cómo proceder para generar código o datos. El archivo objeto resultante es compatible solo con la arquitectura destino.

Ensamblador

El **ensamblador** transforma instrucciones ASM en código binario (código objeto).

Esta transformación es directa, no requiere interpretar qué acciones realiza el programa.

Cuenta con **pseudoinstrucciones** que le indican al ensamblador cómo proceder para generar código o datos. El archivo objeto resultante es compatible solo con la arquitectura destino.

Archivo objeto

Se puede ver como código binario con faltantes e información adicional.

→ Los faltantes son funciones o procedimientos no registrados en el archivo fuente.

→ La información adicional es una **tabla de símbolos**, con todos los símbolos declarados en el archivo objeto, y todos los faltantes con su nombre y dirección relativa.

Tabla de símbolos

A medida que se construye un programa se **genera una tabla con *símbolos***.

Llamamos símbolo a un **nombre** asociado a un **valor** numérico entero.
Cada símbolo puede estar definido (T) o no definido (U), entre otros estados.

Los símbolos asocian nombres a direcciones de memoria **relativas**
dentro de un segmento del archivo objeto.

Tabla de símbolos

A medida que se construye un programa se **genera una tabla con *símbolos***.

Llamamos símbolo a un **nombre** asociado a un **valor** numérico entero.
Cada símbolo puede estar definido (T) o no definido (U), entre otros estados.

Los símbolos asocian nombres a direcciones de memoria **relativas**
dentro de un segmento del archivo objeto.

Ejemplo: Símbolos dentro de un archivo objeto.

```
0000000000000000 T addFirst
                 U free
                 U _GLOBAL_OFFSET_TABLE_
00000000000000d1 T main
                 U malloc
000000000000003c T removeLast
```

Comando útil

nm: Mostrar tabla de símbolos.

Linker

El **Linker** se encarga de tomar un conjunto de archivos objeto y enlazarlos en uno solo.

Resuelve todas las **dependencias** entre archivos, con el sistema y con bibliotecas externas de funciones.

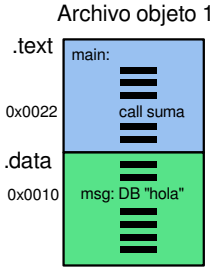
El resultado es un **programa ejecutable** para un sistema operativo particular.

Linker

El **Linker** se encarga de tomar un conjunto de archivos objeto y enlazarlos en uno solo.

Resuelve todas las **dependencias** entre archivos, con el sistema y con bibliotecas externas de funciones.

El resultado es un **programa ejecutable** para un sistema operativo particular.

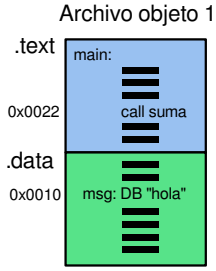


Linker

El **Linker** se encarga de tomar un conjunto de archivos objeto y enlazarlos en uno solo.

Resuelve todas las **dependencias** entre archivos, con el sistema y con bibliotecas externas de funciones.

El resultado es un **programa ejecutable** para un sistema operativo particular.

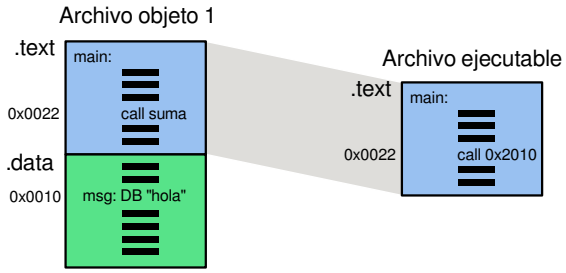


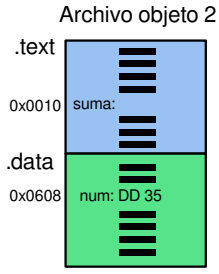
Linker

El **Linker** se encarga de tomar un conjunto de archivos objeto y enlazarlos en uno solo.

Resuelve todas las **dependencias** entre archivos, con el sistema y con bibliotecas externas de funciones.

El resultado es un **programa ejecutable** para un sistema operativo particular.



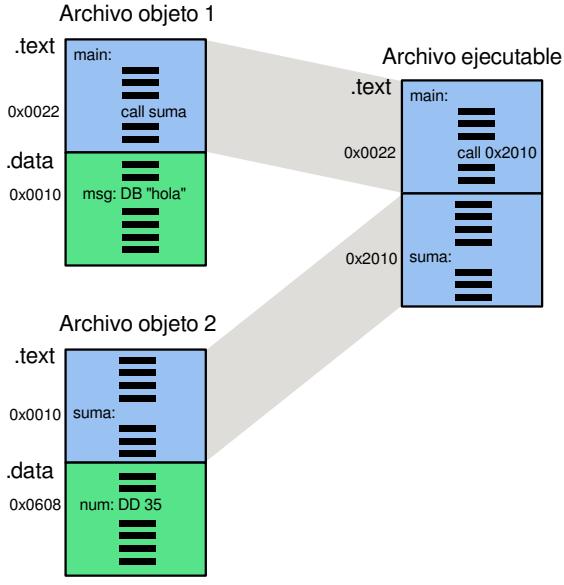


Linker

El **Linker** se encarga de tomar un conjunto de archivos objeto y enlazarlos en uno solo.

Resuelve todas las **dependencias** entre archivos, con el sistema y con bibliotecas externas de funciones.

El resultado es un **programa ejecutable** para un sistema operativo particular.

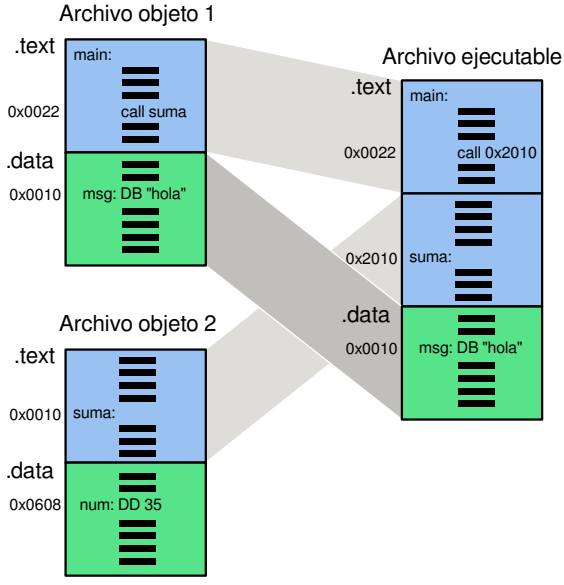


Linker

El **Linker** se encarga de tomar un conjunto de archivos objeto y enlazarlos en uno solo.

Resuelve todas las **dependencias** entre archivos, con el sistema y con bibliotecas externas de funciones.

El resultado es un **programa ejecutable** para un sistema operativo particular.

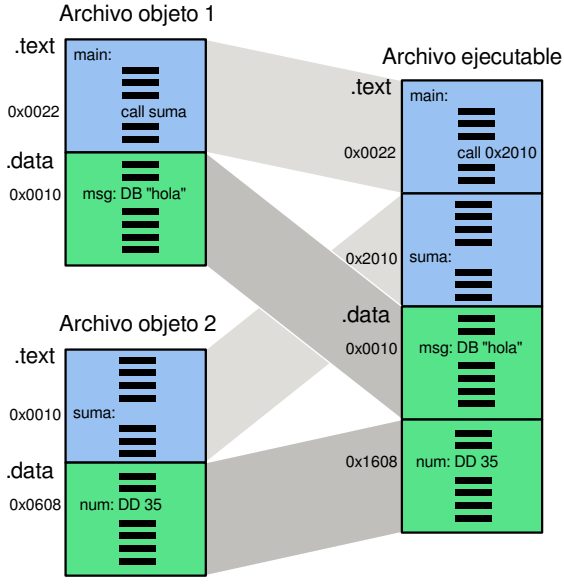


Linker

El **Linker** se encarga de tomar un conjunto de archivos objeto y enlazarlos en uno solo.

Resuelve todas las **dependencias** entre archivos, con el sistema y con bibliotecas externas de funciones.

El resultado es un **programa ejecutable** para un sistema operativo particular.

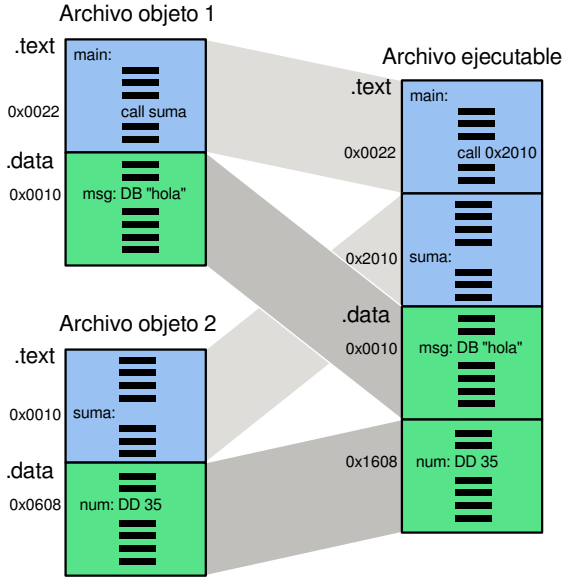


Linker

El **Linker** se encarga de tomar un conjunto de archivos objeto y enlazarlos en uno solo.

Resuelve todas las **dependencias** entre archivos, con el sistema y con bibliotecas externas de funciones.

El resultado es un **programa ejecutable** para un sistema operativo particular.



Comando útil

ldd: Mostrar bibliotecas compartidas (*shared libraries*)

Static vs Dynamic Linker

Las aplicaciones utilizan **bibliotecas** (*library*) de funciones externas. Ejemplo: `libc`.
Estas pueden estar enlazadas de forma estática o dinámica.

Static vs Dynamic Linker

Las aplicaciones utilizan **bibliotecas** (*library*) de funciones externas. Ejemplo: `libc`.
Estas pueden estar enlazadas de forma estática o dinámica.

Enlazado estático

Se resuelve en tiempo de enlazado o linkeo: todo el código de la biblioteca forma parte del binario resultante. Existirán múltiples copias del mismo código en memoria para cada aplicación.

Static vs Dynamic Linker

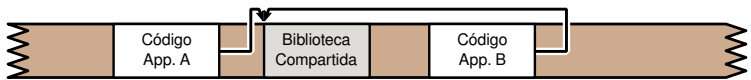
Las aplicaciones utilizan **bibliotecas** (*library*) de funciones externas. Ejemplo: `libc`.
Estas pueden estar enlazadas de forma estática o dinámica.

Enlazado estático

Se resuelve en tiempo de enlazado o linkeo: todo el código de la biblioteca forma parte del binario resultante. Existirán múltiples copias del mismo código en memoria para cada aplicación.

Enlazado dinámico

Se resuelve en tiempo de ejecución: el código de la biblioteca no forma parte del binario. Existe solo una copia de la biblioteca que es usada por múltiples aplicaciones.



Niveles de optimización

La optimización de código se encarga transformar el código intermedio en una nueva versión que mejore alguna característica particular:

-O0 Reduce el tiempo de compilación y permite obtener resultados correctos del debugger.

-O1 Optimiza el costo de ejecución y el tamaño del código, no realiza optimizaciones que demoren mucho tiempo.

-O2 Optimiza más. Agrega todas las optimizaciones, excepto las que compiten con el tamaño del código y el tiempo de ejecución.

-O3 Optimiza más aún. Agrega todas las optimizaciones.

-Os Optimiza el tamaño del ejecutable.

Información de debug

El código cuando es cargado consiste en solo instrucciones y datos.

Para **entender** el flujo del programa y su relación con el código fuente,
es necesario que al momento de compilación se **genere información de *debug***,
y que esta información se preserve durante todo el proceso de generación de código.

Información de debug

El código cuando es cargado consiste en solo instrucciones y datos.

Para **entender** el flujo del programa y su relación con el código fuente,
es necesario que al momento de compilación se **genere información de *debug***,
y que esta información se preserve durante todo el proceso de generación de código.

Ejemplo: Archivo ASM con información de debugging.

```
addFirst:
.LFB6:
    .file 1 "lista.c"
    .loc 1 9 49
    .cfi_startproc
    endbr64
    pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq %rsp, %rbp
```

Compilar, Ensamblar y Enlazar en C

Compilar

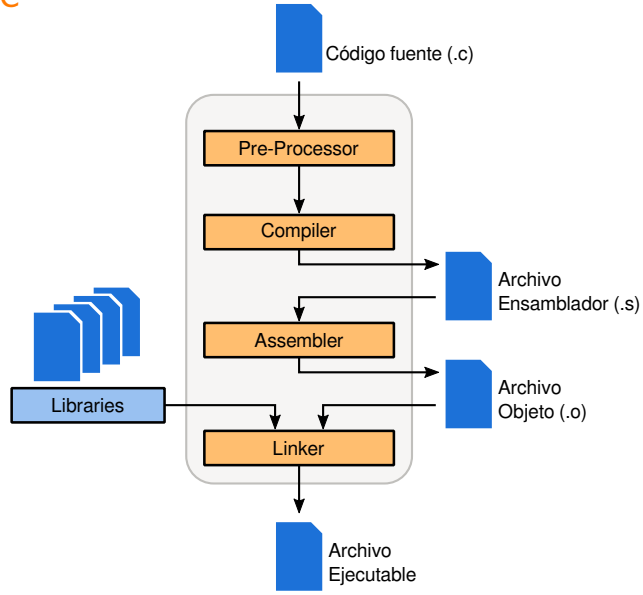
```
$ gcc -S -o lista.s lista.c
```

Ensamblar

```
$ gcc -c -o lista.o lista.c
```

Enlazar

```
$ gcc -o lista.out lista.c
```



Bibliografía

- Tanenbaum, “Organización de Computadoras. Un Enfoque Estructurado”, 4ta Edición, 2000.
  - **Capitulo 7 - El nivel de lenguaje ensamblado**
    - 7.3 El proceso de ensamblado
    - 7.4 Enlazado y carga

- Null, “Essentials of Computer Organization and Architecture”, 5th Edition, 2018.
  - **Chapter 8 - System Software**
    - 8.4.1 Assemblers and Assembly
    - 8.4.2 Link Editors
    - 8.4.3 Dynamic Link Libraries
    - 8.4.4 Compilers
    - 8.4.5 Interpreters

¡Gracias!

Recuerden leer los comentarios adjuntos
en cada clase por aclaraciones.