Facultad de Ingeniería en Electricidad y Computación

Programación de Sistemas CCPG1051

Federico Domínguez, PhD.

Unidad 2 – Sesión 1: Compiladores

Agenda

1. Sistemas de Compilación
2. Arquitectura y representación de datos
3. GNU Compiler Collection

Sistemas de Compilación

¿Qué es un compilador?

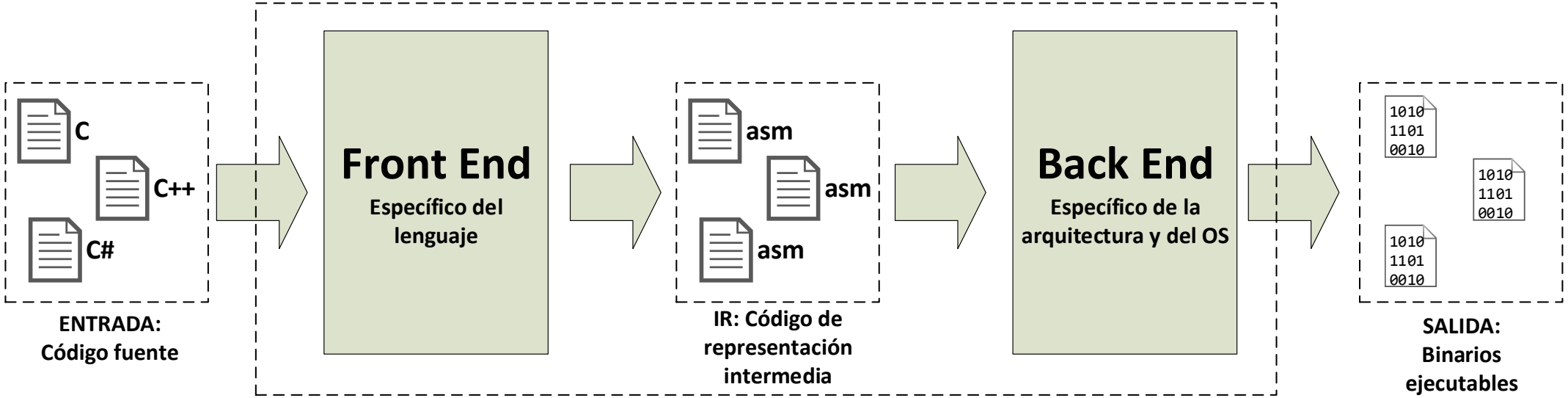
*"Un **compilador** es un programa informático que **traduce** un programa que ha sido escrito en un lenguaje de programación a un lenguaje diferente, usualmente **lenguaje de máquina** … Este proceso de traducción se conoce como compilación."*

– Wikipedia en Español

Un compilador es usualmente parte de una cadena de herramientas, un **sistema de compilación.**

Un sistema de compilación es un grupo de herramientas de software las cuales en conjunto producen la traducción de **lenguaje de alto nivel** a **lenguaje de máquina**. Conocido como:

- Compiler System
- Compiler toolchain



Existen un sinnúmero de **sistemas de compilación** para sistemas embebidos propietarios; para sistemas de uso masivo como **Linux** existen básicamente dos (código abierto).

GCC: GNU Compiler Collection

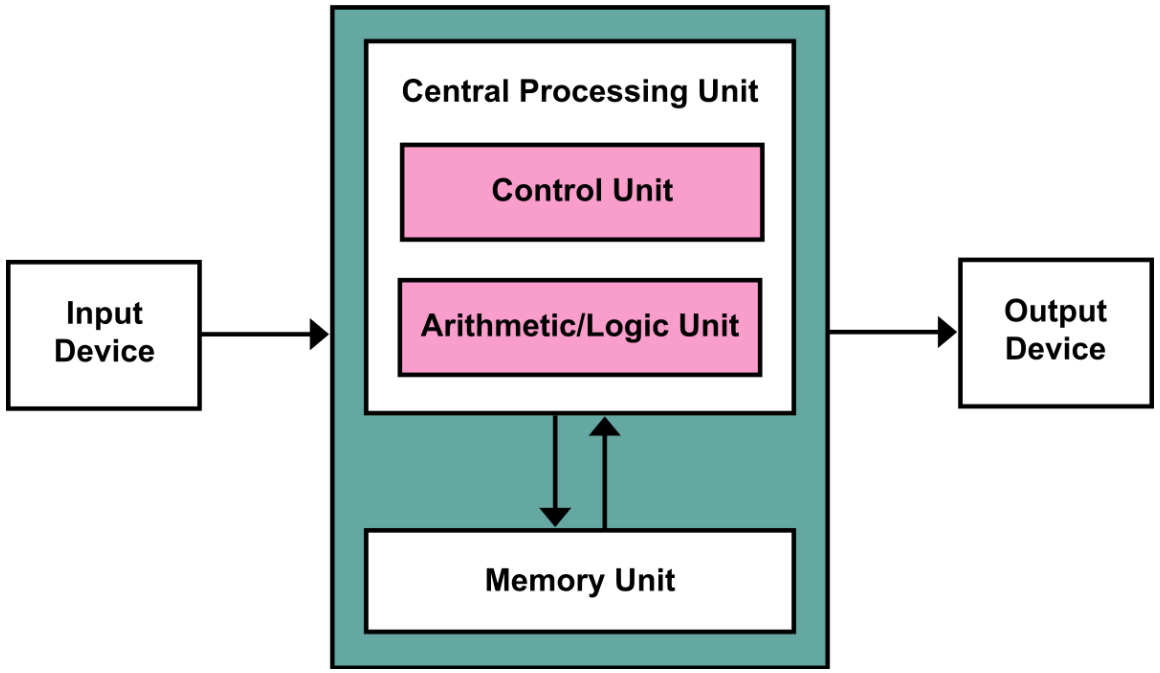
- Distribuciones de Linux y FreeBSD, código abierto
- gcc: GNU C Compiler (nombre antiguo de GCC)
- g++: GNU C++ Compiler

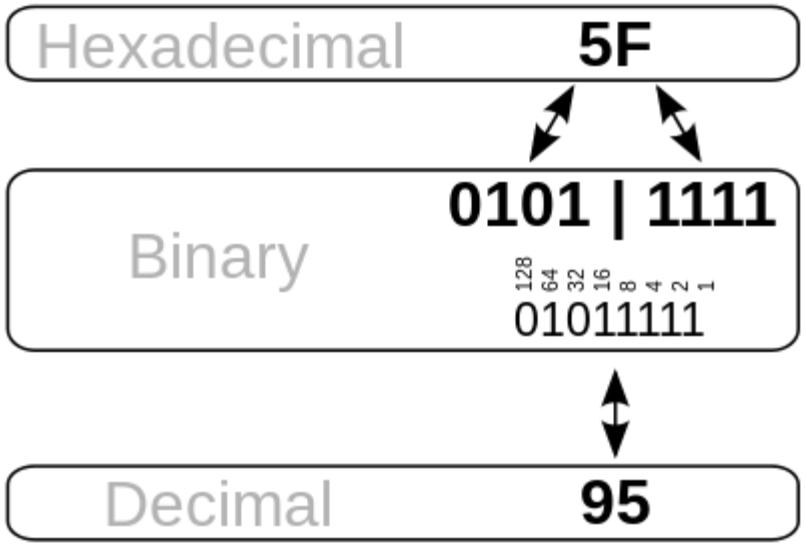
LLVM: Low Level Virtual Machine (acrónimo obsoleto)

- macOS, algunas distribuciones de UNIX, opcional en LINUX, código abierto
- Clang: Compilador de C, C++, Objective-C, Objective-C++, OpenCL, CUDA …
- Diseñado para reemplazar a GCC

.Net SDK: Equivalente para desarrollo de aplicaciones en Microsoft Windows

Arquitecturas y representación de datos





Arquitecturas de Procesador

PowerPC: Arquitectura basada en un set de instrucciones RISC. Desarrollado por Apple, IBM y Motorola. Usado por todos los productos de Apple hasta 2006.

ARM: **A**dvanced **R**ISC **M**achine, desarrollado por la compañía británica ARM Holdings. Debido al uso de un set de instrucciones RISC y a un diseño eficiente, es ampliamente usado en sistemas embebidos. En términos de cantidad, es la arquitectura más usada (100 mil millones de procesadores).

x86: Creada por Intel en 1985 para los procesadores de la línea 80x86.

- Procesadores 8086 y 80286: 16 bits, usados en las primeras PCs de IBM.
- **i386:** Primer procesador Intel de 32 bits. El set de instrucciones de este procesador es conocido también como **IA32** (**I**ntel **A**rchitecture, **32**-bit**).**
- **i486, Pentium, PentiumPro, Pentium II, Pentium III, Pentium 4:** Mejora acumulada de la arquitectura IA32.
- **Pentium 4E: AMD (A**dvanced **M**icro **D**evices**)** en 2004 introduce una extensión de 64 bits a la arquitectura IA32 conocida ahora como **Intel64** o **x86-64**. También se introduce *hyperthreading* en este procesador.
- **Core 2:** Primer procesador de Intel **multi-core** introducido en 2006.
- **Core i3, i5, i7:** Procesadores multi-core con soporte *hyperthreading*.
- x86 y x86-64 dominan el mercado de computadores personales y servidores.

El **código de máquina** es específico a la **arquitectura** porque se exponen variables internas que representan el estado del procesador.

Código de máquina usa:

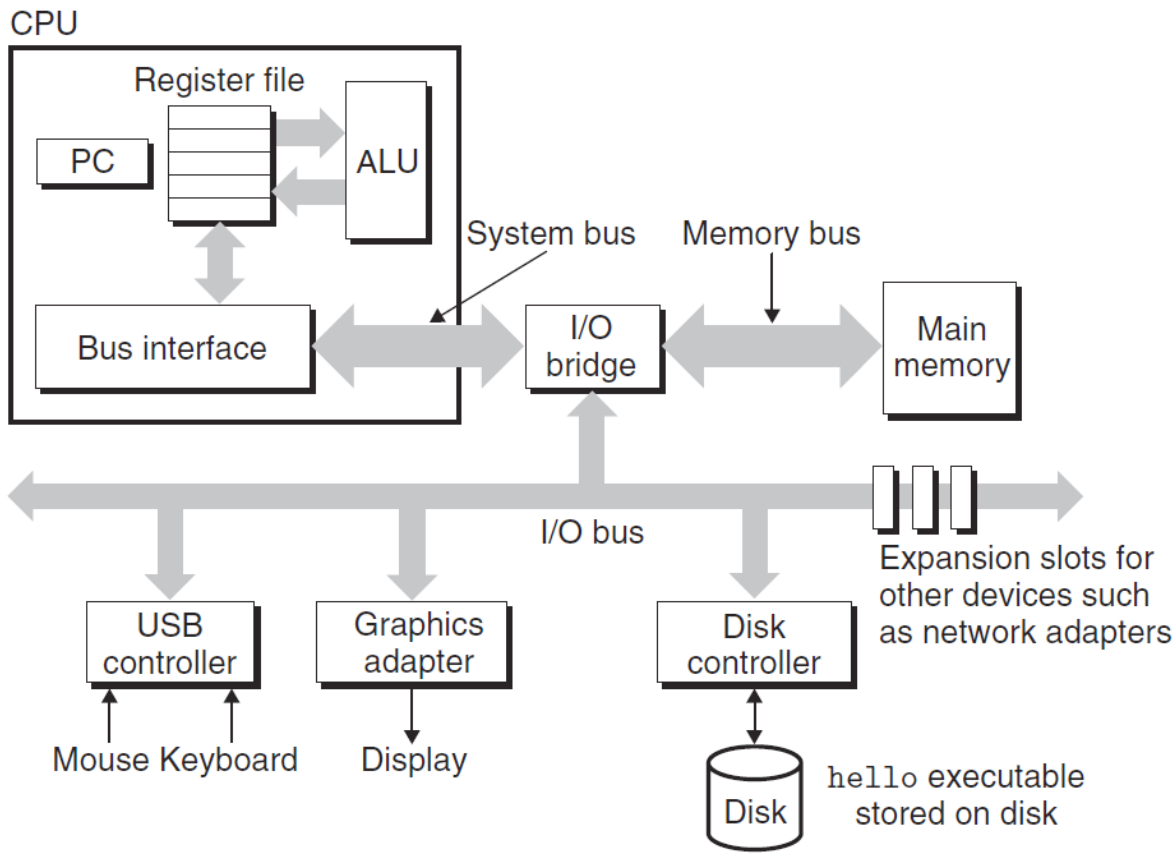
direccionamiento virtual, en x86-64 cada dirección de memoria es de 64 bits,

directamente los registros del procesador, 16 locaciones de 64 bits con nombres específicos,

directamente el registro especial *program counter* (PC),

registros especiales condicionales,

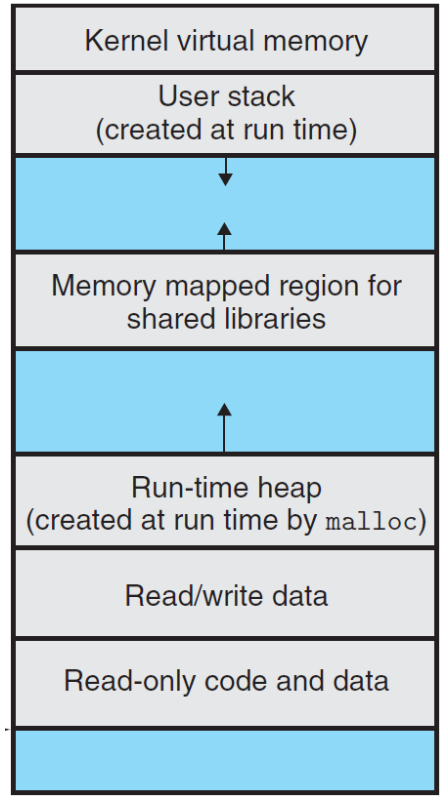
registros vectoriales para operaciones de punto flotante.



El código de máquina es específico en el direccionamiento de memoria virtual.

En código de maquina se usa directamente el ***stack*** (o pila LIFO) del proceso para mantener los parámetros de las funciones y los retornos de las funciones.

Una sección de memoria llamada ***heap*** se usa además para uso de memoria dinámica.



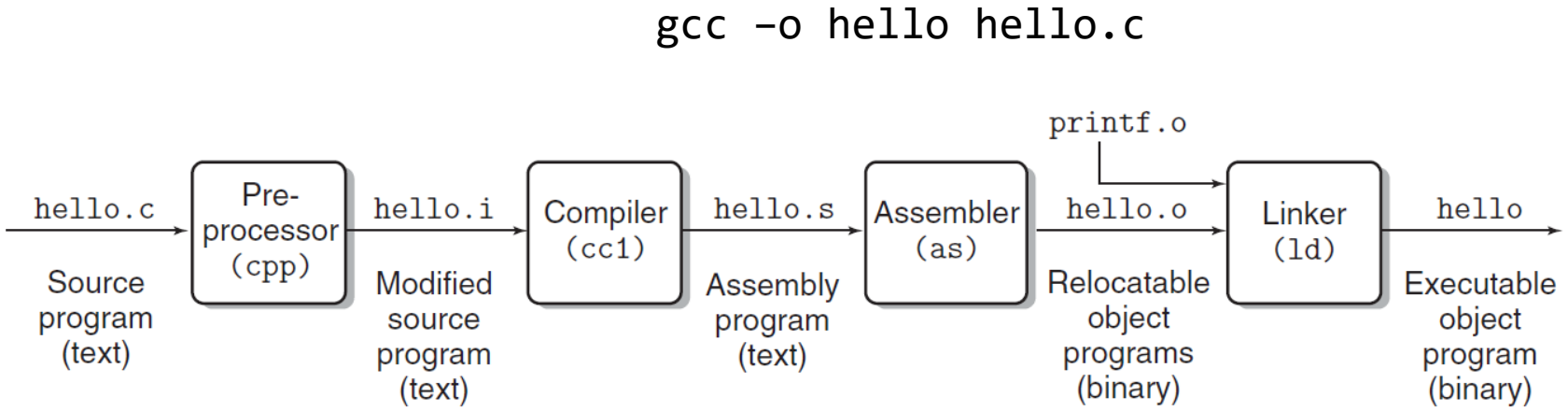
Representación de datos en arquitectura x86-64

Declaración en C	Tipo de datos en Intel64	Tamaño (bytes)
char	Byte	1
short	Word	2
int	Double Word	4
long	Quad Word	8
char *	Quad Word	8
float	Single precision	4
double	Double precision	8

GNU Compiler Collection

GNU Compiler Collection
Compiler driver

El comando **gcc** es en realidad un **compiler driver**, un programa que se encarga de ejecutar toda la cadena de programas que forman GNU Compiler Collection.

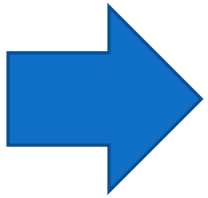


Compilando con gcc –Og podemos estudiar la conversión de lenguaje de C a lenguaje de máquina sin optimizaciones.

En una máquina IA32:
`gcc -Og -S code.c`

code.c

```
1    int accum = 0;
2
3    int sum(int x, int y)
4    {
5        int t = x + y;
6        accum += t;
7        return t;
8    }
```



code.s

```
sum:
    pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    addl    8(%ebp), %eax
    addl    %eax, accum
    popl    %ebp
    ret
```

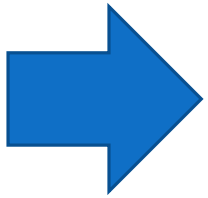
Compilando con gcc –Og podemos estudiar la conversión de lenguaje de C a lenguaje de máquina sin optimizaciones.

En una máquina IA32:

```
gcc –Og –c code.c
```

code.c

```
1    int accum = 0;
2
3    int sum(int x, int y)
4    {
5        int t = x + y;
6        accum += t;
7        return t;
8    }
```



code.o

```
1    00000000 <sum>:
     Offset   Bytes
2       0:    55
3       1:    89 e5
4       3:    8b 45 0c
5       6:    03 45 08
6       9:    01 05 00 00 00 00
7       f:    5d
8      10:    c3
```

Existe un mapeo directo entre código de máquina y *assembler*.

Se puede regresar de código de máquina a *assembler* con un programa como objdump.

```
objdump -d code.o
```

```
1  Disassembly of function sum in binary file code.o
   00000000 <sum>:
   Offset   Bytes              Equivalent assembly language
2      0:   55                 push   %ebp
3      1:   89 e5              mov    %esp,%ebp
4      3:   8b 45 0c           mov    0xc(%ebp),%eax
5      6:   03 45 08           add    0x8(%ebp),%eax
6      9:   01 05 00 00 00 00  add    %eax,0x0
7      f:   5d                 pop    %ebp
8     10:   c3                 ret
```

Esto lo resuelve el *linker*

Demostración

Linking es el proceso de recolectar y combinar varios archivos de código en un archivo único ejecutable.

Linking permite la compilación de programas usando varios archivos. Indispensable en proyectos medianos y grandes.

Comprender el proceso de *linking* es importante cuando se esta construyendo programas de complejidad alta.

El alcance de las variables está relacionado al proceso de *linking*.

- La diferencia entre variables globales y locales.
- La palabra clave **`static`** en C esta relacionada al proceso de *linking*. No es lo mismo que **`static`** en Java.

Linking estático es hecho en tiempo de compilación por el programa **`ld`**.

El proceso de *linking* toma como entradas *object files* (.o). Archivos en binario con direcciones de memoria relativas.

En la compilación estática, el programa **ld** debe hacer dos tareas:

1. *Symbol resolution*. Un símbolo corresponde a una **función**, **variable global** o una **variable local** declarada con `static`. En este paso, cada símbolo es reemplazado por una referencia única en toda la compilación.
2. *Relocation*. Compiladores generan *object files* con sus secciones de código y datos empezando en la dirección de memoria 0. En este paso, el *linker* reemplaza estas direcciones con direcciones virtuales validas en tiempo de ejecución.

Linking estático se encarga de identificar los símbolos que deben de ser resueltos.

(a) `main.c`

code/link/main.c

```
1   /* main.c */
2   void swap();
3
4   int buf[2] = {1, 2};
5
6   int main()
7   {
8       swap();
9       return 0;
10  }
```

code/link/main.c

Símbolo resuelto por el *linker*

(b) `swap.c`

code/link/swap.c

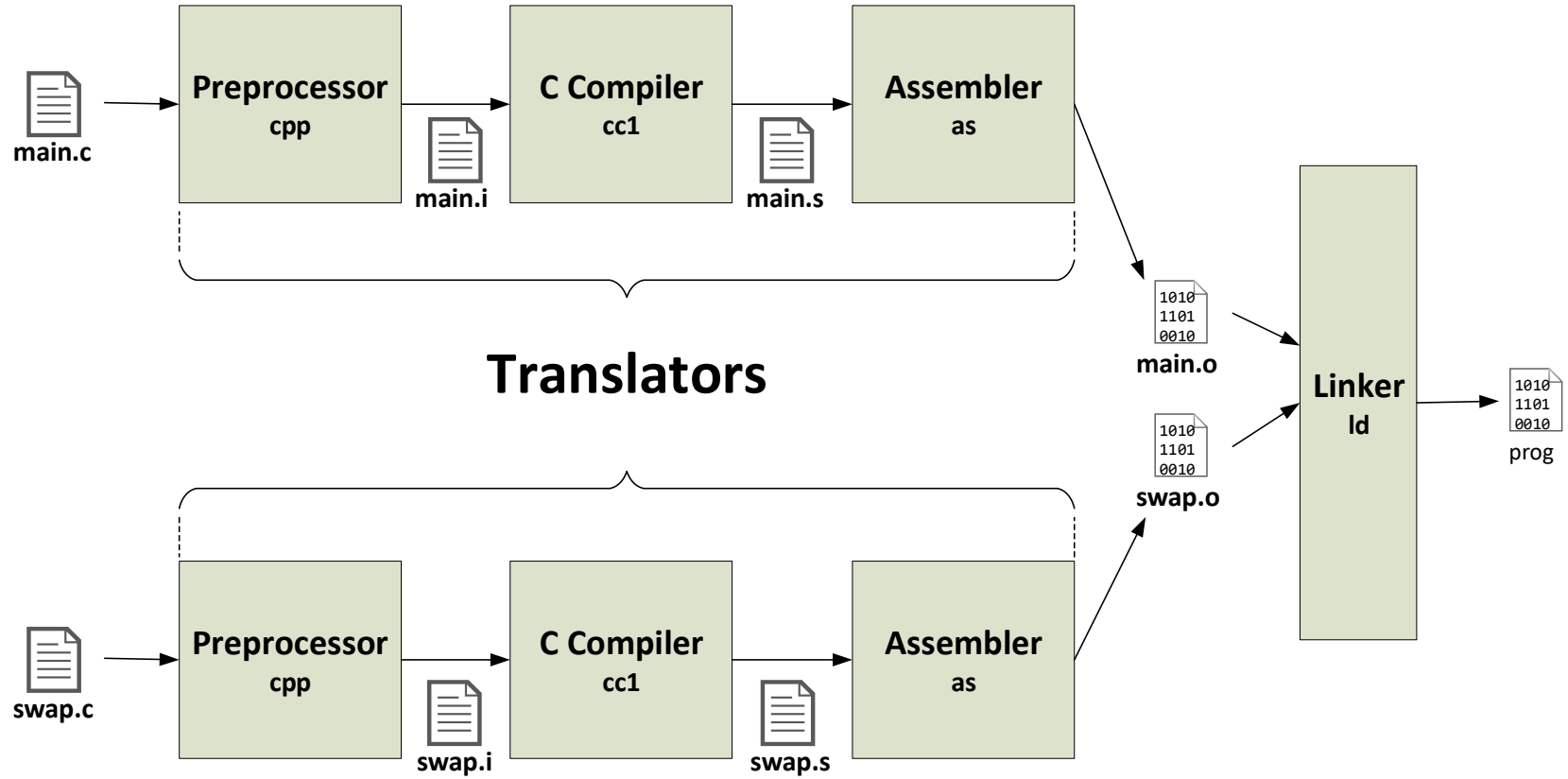
```
1   /* swap.c */
2   extern int buf[];
3
4   int *bufp0 = &buf[0];
5   int *bufp1;
6
7   void swap()
8   {
9       int temp;
10
11      bufp1 = &buf[1];
12      temp = *bufp0;
13      *bufp0 = *bufp1;
14      *bufp1 = temp;
15  }
```

code/link/swap.c

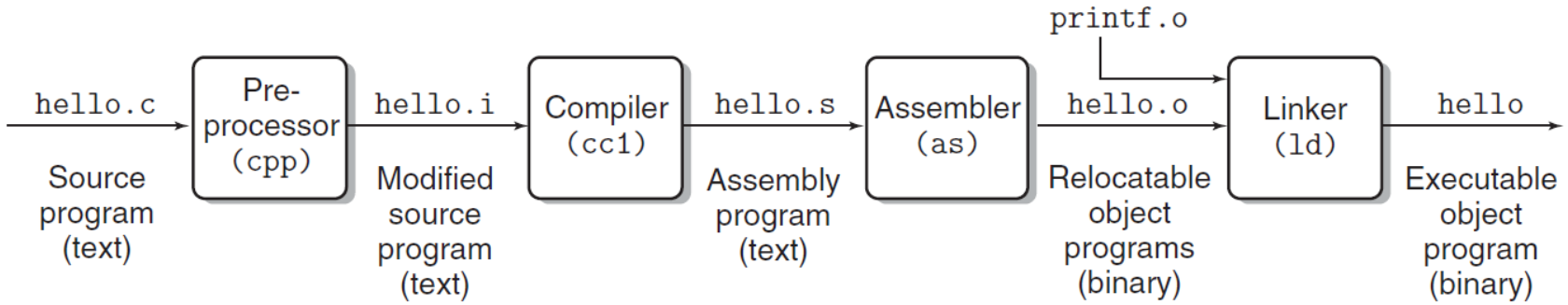
Símbolo resuelto por el *linker*

El proceso de *linking* une la compilación de `main.c` y `swap.c` en un solo programa ejecutable.

```
gcc –o prog main.c swap.c
```



En resumen



El sistema de compilación en GCC consta de cuatro fases:

1. **Pre-procesamiento**: Reemplazar directivas de compilación (empiezan con #). Por ejemplo `#include` o `#define`.
2. **Compilación**: Compilar C a ensamblador.
3. **Ensamblado**: Convertir ensamblador a lenguaje de máquina.
4. **Linking**: Fusionar diferentes archivos en lenguaje de máquina a un solo archivo ejecutable.

Demostración

Referencia

Libro Computer Systems, Bryant y O'Hallaron. Secciones 1.1 – 1.4, 3.1 – 3.3 y 7.1 – 7.4