Herramientas de creación y carga de ejecutables

EJEMPLO DE LA SUMA DE CUADRADOS DE 0 A 100

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Lenguaje Máquina MIPS

```
addiu   $29, $29, -32
sw      $31, 20($29)
sw      $4,  32($29)
sw      $5,  36($29)
sw      $0,  24($29)
sw      $0,  28($29)
lw      $14, 28($29)
lw      $24, 24($29)
multu   $14, $14
addiu   $8,  $14, 1
slti    $1,  $8, 101
sw      $8,  28($29)
mflo    $15
addu    $25, $24, $15
bne     $1,  $0, -9
sw      $25, 24($29)
lui     $4,  4096
lw      $5,  24($29)
jal     1048812
addiu   $4,  $4, 1072
lw      $31, 20($29)
addiu   $29, $29, 32
jr      $31
move    $2,  $0
```

Ensamblador

```
        .text
        .align  2
        .globl  main
main:
        subu    $sp, $sp, 32
        sw      $ra, 20($sp)
        sd      $a0, 32($sp)
        sw      $0,  24($sp)
        sw      $0,  28($sp)
loop:
        lw      $t6, 28($sp)
        mul     $t7, $t6, $t6
        lw      $t8, 24($sp)
        addu    $t9, $t8, $t7
        sw      $t9, 24($sp)
        addu    $t0, $t6, 1
        sw      $t0, 28($sp)
        ble     $t0, 100, loop
        la      $a0, str
        lw      $a1, 24($sp)
        jal     printf
        move    $v0, $0
        lw      $ra, 20($sp)
        addu    $sp, $sp, 32
        jr      $ra

        .data
        .align  0
str:
        .asciiz "The sum from 0 .. 100 is %d\n"
```

Ensamblador con Etiquetas

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
    int i;
    int sum = 0;

    for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
    printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Lenguaje C

Diagrama 2: Posibles modos de funcionamiento de un compilador para un lenguaje de alto nivel.

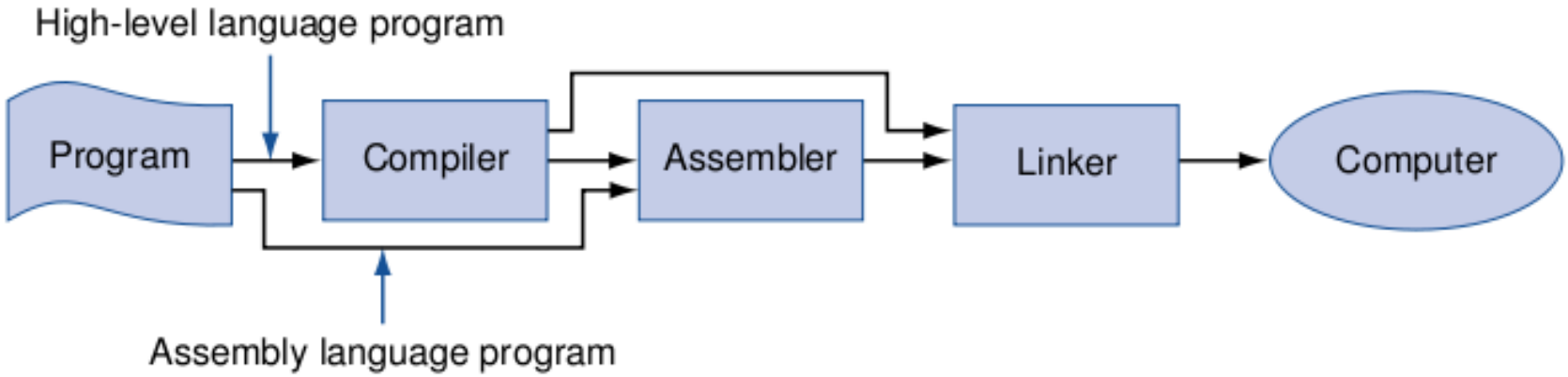


Diagrama 1: Proceso de construcción de un fichero ejecutable mediante la herramienta "ensamblador" y "enlazador" o linker.

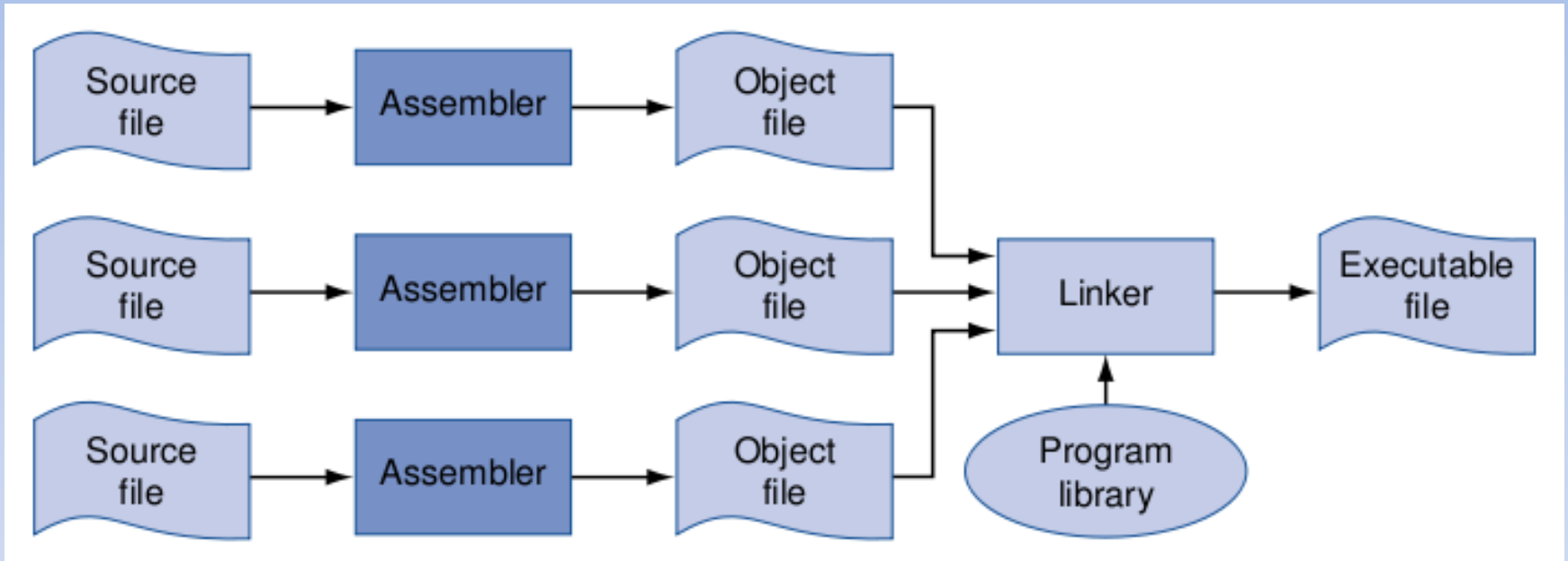
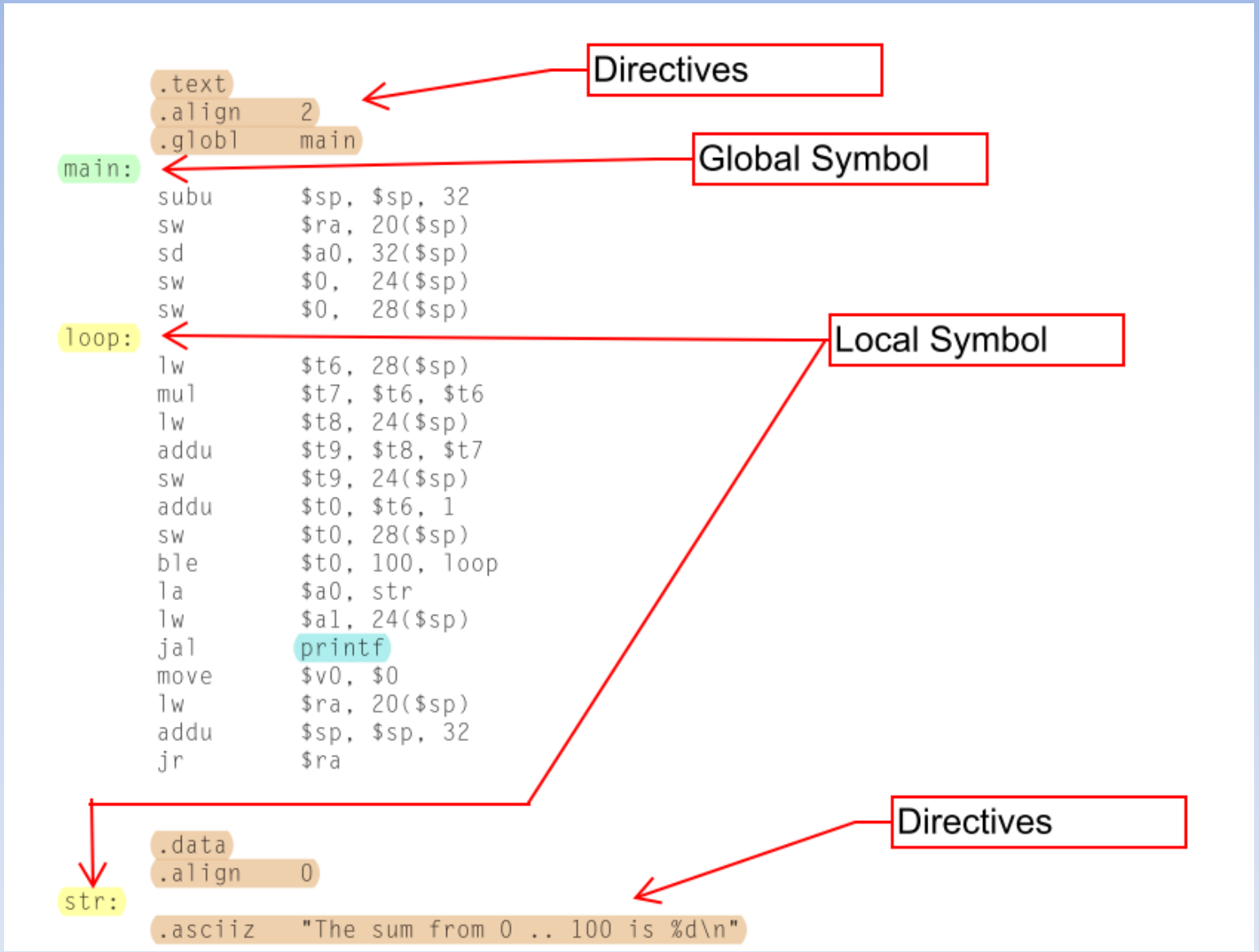


Diagrama 1: Proceso de construcción de un fichero ejecutable mediante la herramienta "ensamblador" y "enlazador" o linker.

Labels, Symbols and Directives



Etiquetas, Directivas y Macros

- **Etiqueta**: Nombre que identifica una dirección de código. Por ejemplo "**loop**"
  - **Local**: Si el objeto se puede utilizar solamente dentro del fichero en el que está definida. Es la situación por defecto. Sus nombres no son visibles desde otros ficheros.
  - **Externa o Global**: Si el objeto nombrado puede ser referenciado desde otros ficheros distintos a aquél en el que se ha definido. ("**main**")
- **Directiva**: Le dice al compilador cómo traducir un programa, pero no producen código. Siempre empiezan con un ".". Por ejemplo "**.data**"
- **Data layout directive**: Describe los datos de forma más concisa y natural que su representación binaria. Por ejemplo ".asciiz"
- **Macro**: Etiqueta que es sustituida por código en el momento del ensamblado. Se utilizan para evitar repetir código frecuentemente

Funcionamiento del ensamblador

- Primera pasada:
  - Lee cada línea del fichero y la separa en cada uno de sus componentes (lexemas)
  - ble | $t0 | , | 100 | , | loop.
  - Guarda las etiquetas y sus direcciones en la tabla de símbolos
- Segunda pasada:
  - Utilizando la tabla de símbolos, lee de nuevo cada línea y si contiene una instrucción produce una instrucción de máquina combinando el código de operación de la instrucción y los operandos (ya sea el identificador del registro o la dirección de memoria correspondiente). Si quedaran referencias no resueltas en la tabla de símbolos, no se queja, ya que probablemente se corresponda con una etiqueta definida en otro fichero.

Object File Format

- The *object file header* describes the size and position of the other pieces of the file.
- The **text segment** contains the machine language code for routines in the source file. These routines may be unexecutable because of unresolved references.
- The **data segment** contains a binary representation of the data in the source file. The data also may be incomplete because of unresolved references to labels in other files.
- The **relocation information** identifies instructions and data words that depend on **absolute addresses.** These references must change if portions of the program are moved in memory.
- The *symbol table* associates addresses with external labels in the source file and lists unresolved references.
- The *debugging information* contains a concise description of the way in which the program was compiled, so a debugger can find which instruction addresses correspond to lines in a source file and print the data structures in readable form.

Symbol Table

Tabla que contiene el nombre y la dirección de cada etiqueta.

Sólo se aplica a los símbolos locales, ya que de los globales no dispone de su dirección.

1. Si es generada por el ensamblador, sólo resuelve las referencias locales. Los símbolos externos no están resueltos
2. Si es generada por el linkador con el propósito de unir varios ficheros objeto, pueden quedar referencias externas no resueltas.
3. Si es generada por el linkador para producir un ejecutable, como el nuevo fichero objeto generado contiene **todo el código**, todas las referencias deben quedar resueltas, ya que no existen referencias externas.

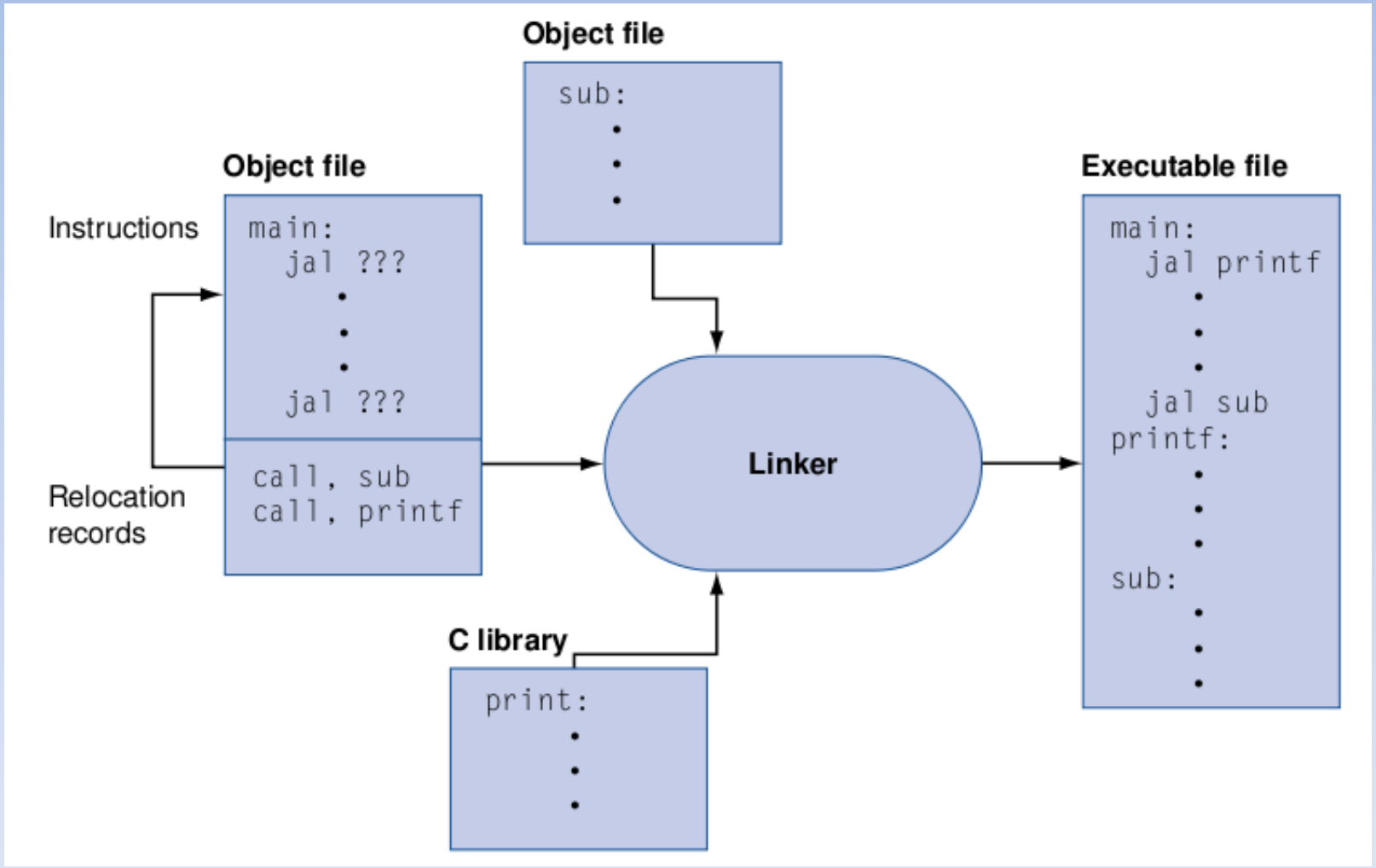
Funcionamiento del Linker (1)

- Un programa se puede dividir en piezas que se almacenan en ficheros diferentes
- Cada fichero se denomina módulo.
- Los ficheros se pueden compilar o ensamblar independientemente unos de otros
- Es necesario realizar un paso que combine todos los ficheros en uno solo.

Funcionamiento del Linker (2)

1. Busca las librerías del programa para encontrar todas las rutinas utilizadas por el programa.
2. Determina las posiciones de memoria que ocupará cada módulo y modifica las referencia dentro de cada módulo produciendo referencia absolutas.
3. Resuelve las referencias entre ficheros (símbolos externos a cada fichero)

Funcionamiento del enlazador (Linker)



Reubicación (Relocation) del linker

- La información de reubicación indica las palabras de datos e instrucciones que dependen de una dirección absoluta. Estas referencias deben cambiar si el programa cambia de posición en la memoria.
- Esta información es necesaria porque el ensamblador no conoce en qué posición de memoria va a quedar un determinado módulo cuando sea enlazado con el resto del programa.
- Cuando el linker coloca un módulo en memoria, todas sus referencias absolutas son reubicadas para reflejar su verdadera posición.

Proceso de carga

- Se debe haber compilado, ensamblado y linkado sin errores.
- El programa se encuentra almacenado en disco duro.

Proceso de carga

- El sistema operativo debe:
  - Leer el programa de disco y determinar el tamaño de los segmentos de datos y código.
  - Crear un nuevo espacio de direcciones para el programa, suficientemente grande para que contenga el segmento de datos, el segmento de código y la pila.
  - Copiar instrucciones y datos desde el disco duro al nuevo espacio de direcciones.
  - Copiar los argumentos que se pasan al programa en la pila
  - Inicializar los registros. Asignar el puntero de pila a la primera dirección libre de la pila.
  - Saltar a la rutina de comienzo que copia los argumentos del programa de la pila a los registros y llama a la rutina “main” del programa.
  - Cuando retorna del “main”, la rutina de comienzo termina el programa con una llamada a “exit system”.