# Introducción

# Presentación de la materia

Objetivos

- ✔ Conocer y programar diferentes tipos de procesadores.
- ✔ Conocer y programar los periféricos de la PC.
- ✔ Programar la familia Intel en bajo nivel.
- ✔ Base de conocimientos para la materia Sistemas Operativos.

# Preguntas para responder en Arqui

¿Como circula y se almacena la información en la PC ?

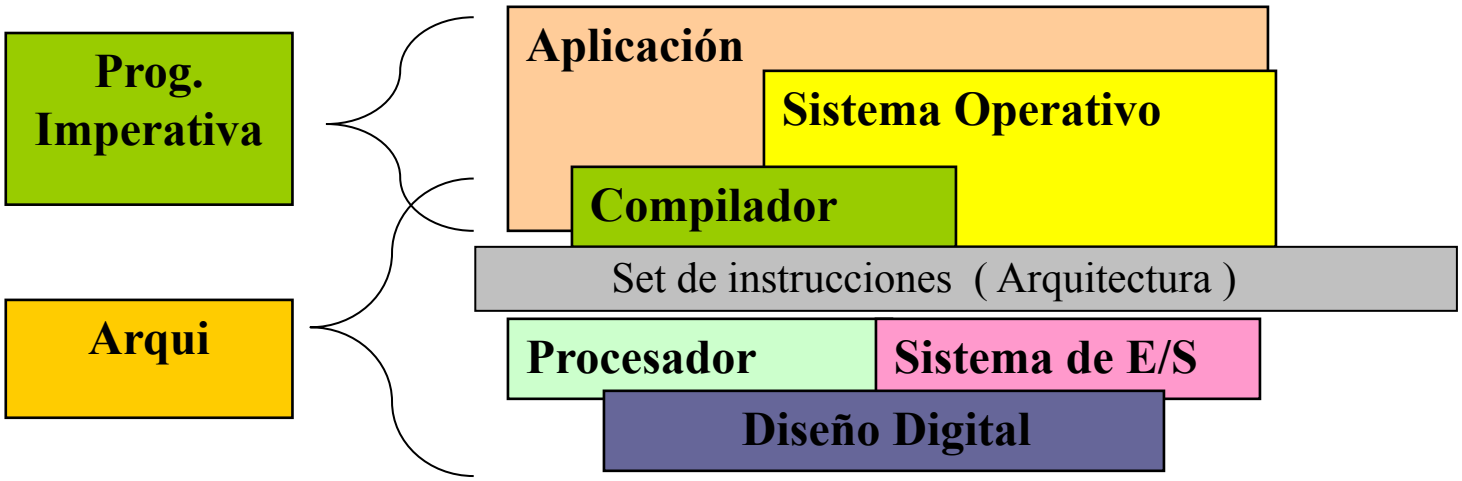
¿Siempre está corriendo el Sistema Operativo ?

¿Como se protegen las aplicaciones?

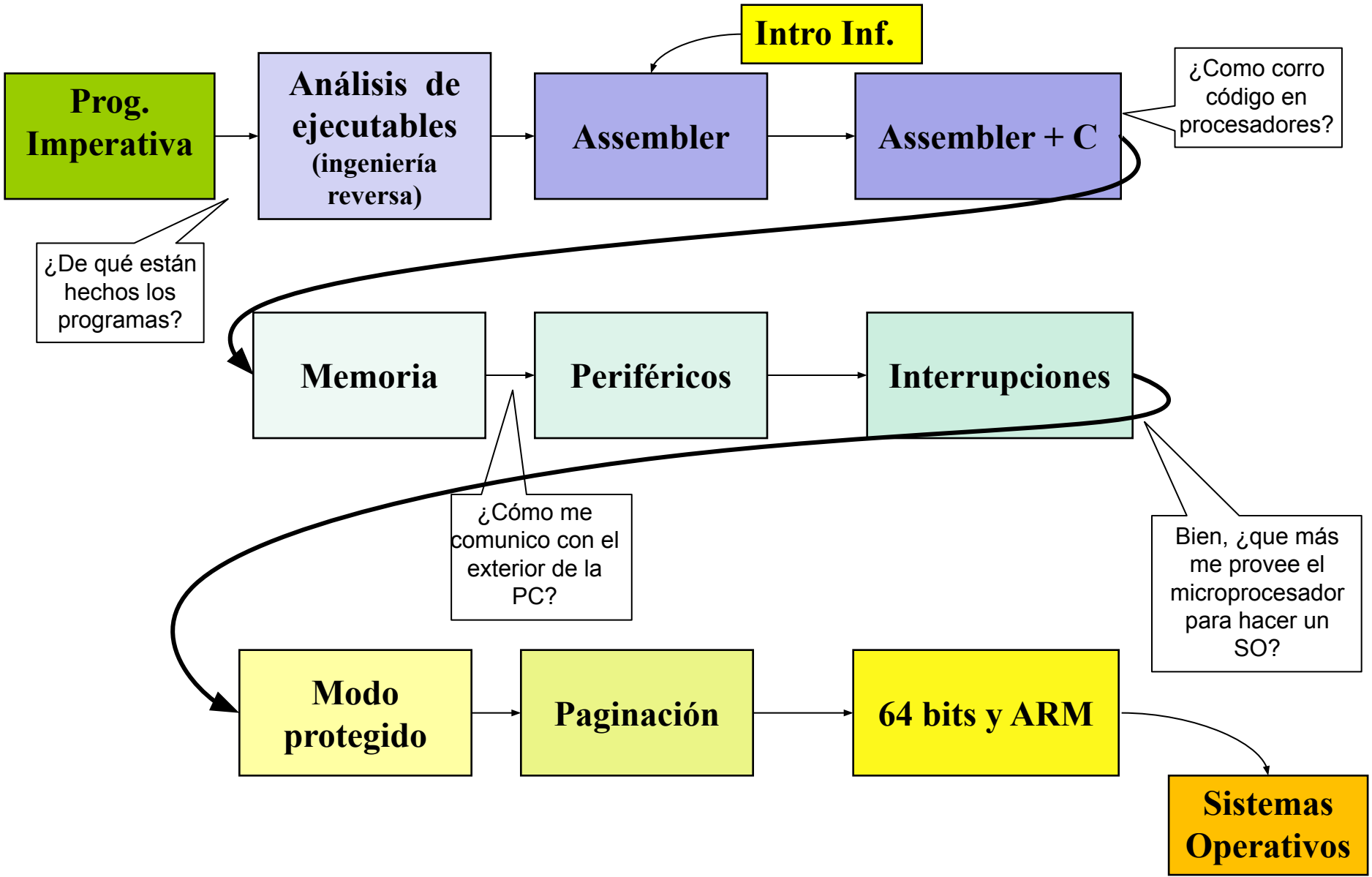
¿Una GPU es mas rapida que una CPU ?

# Presentación de la materia

¿Que capas vemos en cada materia ?



# Presentación de la materia

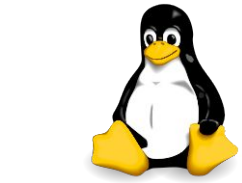


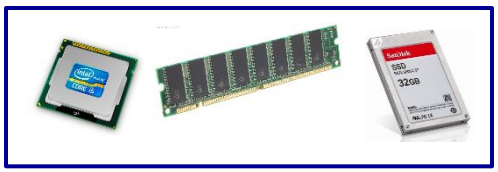
# Bibliografía


- Los microprocesadores Intel. Barry B. Brey. Prentice Hall.
- Organización de Computadoras. Andrew S. Tanenbaum. Prentice Hall
- Organización y Arquitectura de Computadoras. William Stallings. Megabyte.
- **Apuntes de la materia**.

# Arquitecturas

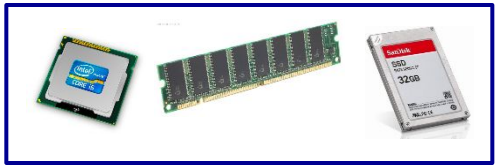
# En la actualidad





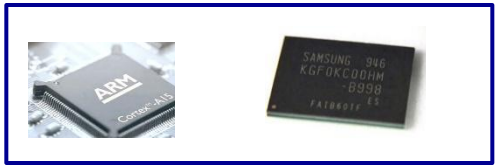
Hardware PC

Variedad de Fabricantes

Hardware Mac

Un fabricante



Hardware Smartphone

Variedad de Fabricantes

# En la actualidad (Consolas)

Xbox One X

Procesador AMD (Jaguar)
Dual Core 64 bits – 4 núcleos c/u

Memoria 12 GB

Bus a memoria 384 bits

PS5

Procesador AMD (Zen 2)
64 bits – 8 núcleos c/u

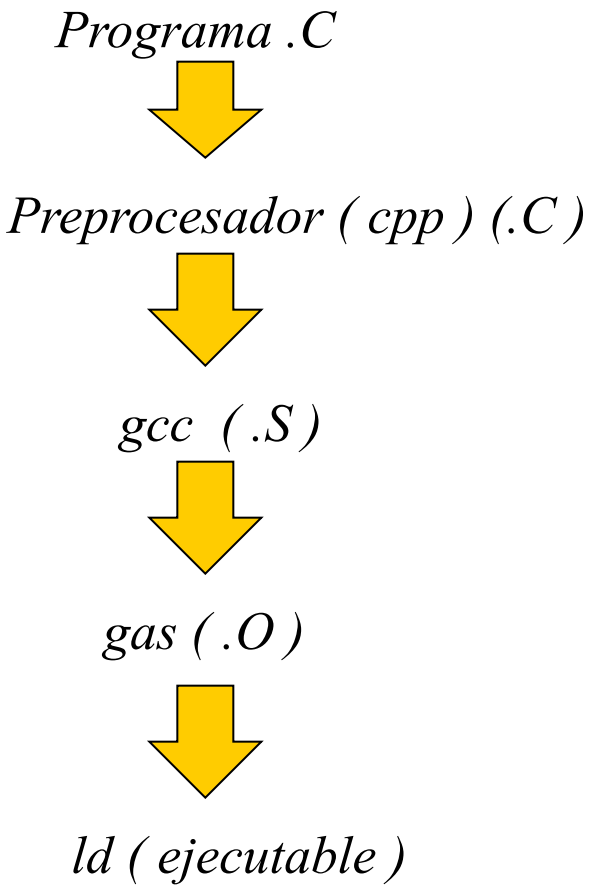
Memoria 16 GB

Bus a memoria 256 bits

# Programas y binarios

# Compilación y linkedición en C (de PI)

Un programa en C al ser compilado sigue el siguiente camino:



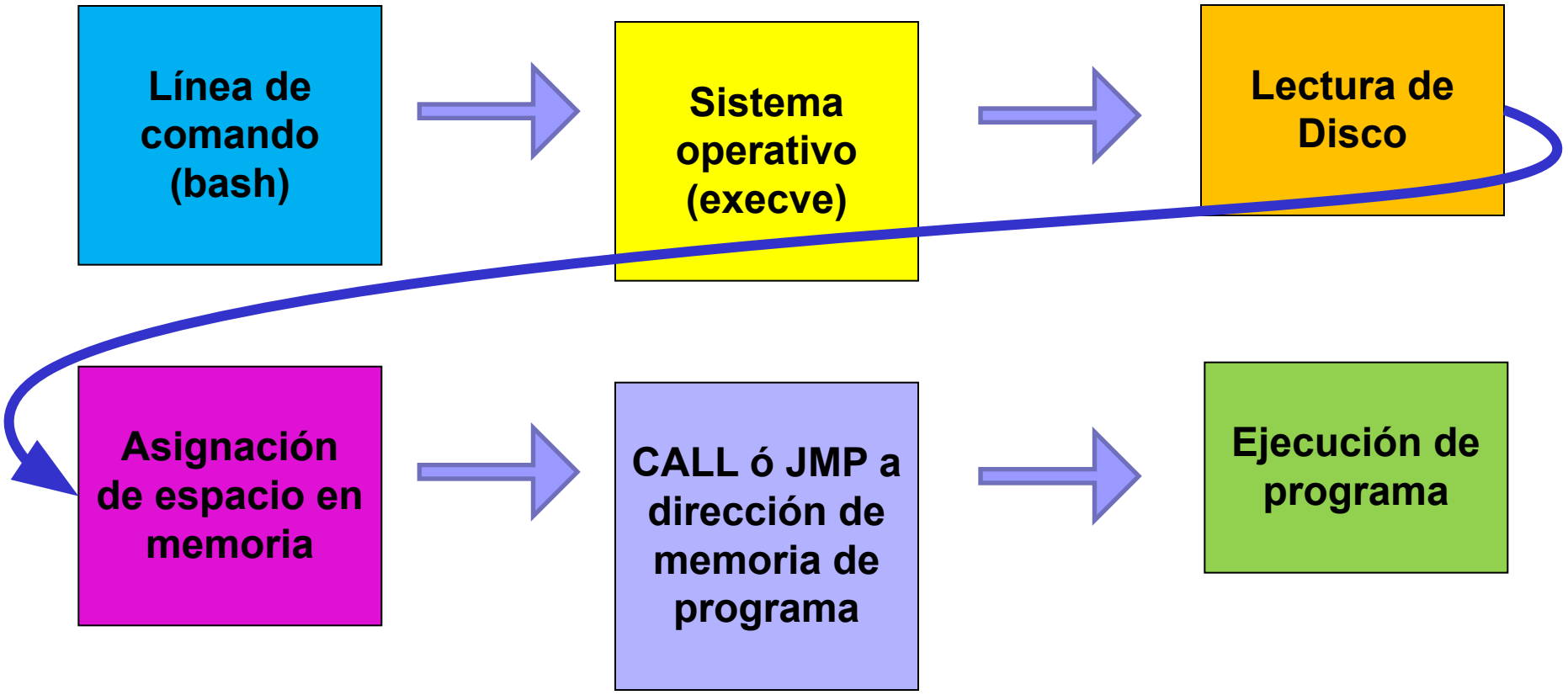

gcc primero invoca al preprocesador cpp, el cual toma el código fuente original y realiza la macroexpansion de directivas como #include y #define.

Luego gcc toma el control de esa salida y convierte el código en C en código equivalente en Assembler. Genera un archivo con extensión ".S". Este código en ASM puede ser generado en sintaxis AT&T ó en la sintaxis Intel

Luego el compilador GNU de Assembler "gas" se encarga de generar el código objeto ( con extensión .O )

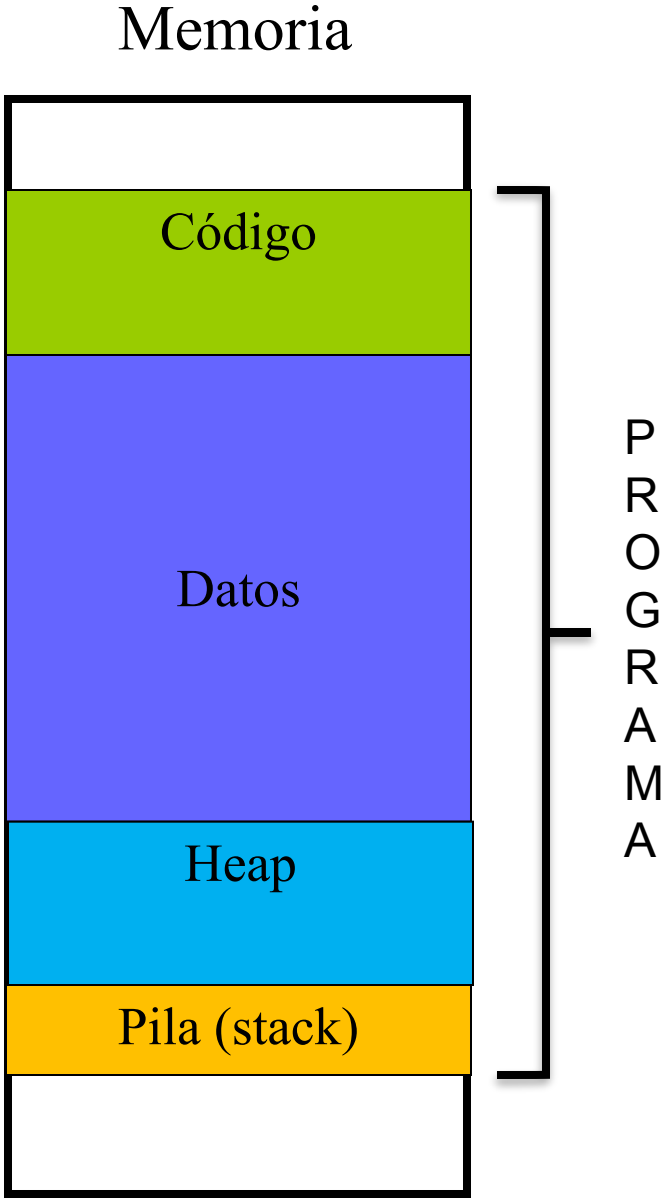
En Linux el ejecutable podrá ser de diferentes formatos, por ej, ELF (mas reciente ) y A.OUT .Se debe tener en cuenta en este punto que el linkeditor "ld" también puede generar el modelo flat , también llamado binario.

# Ejecución de un programa

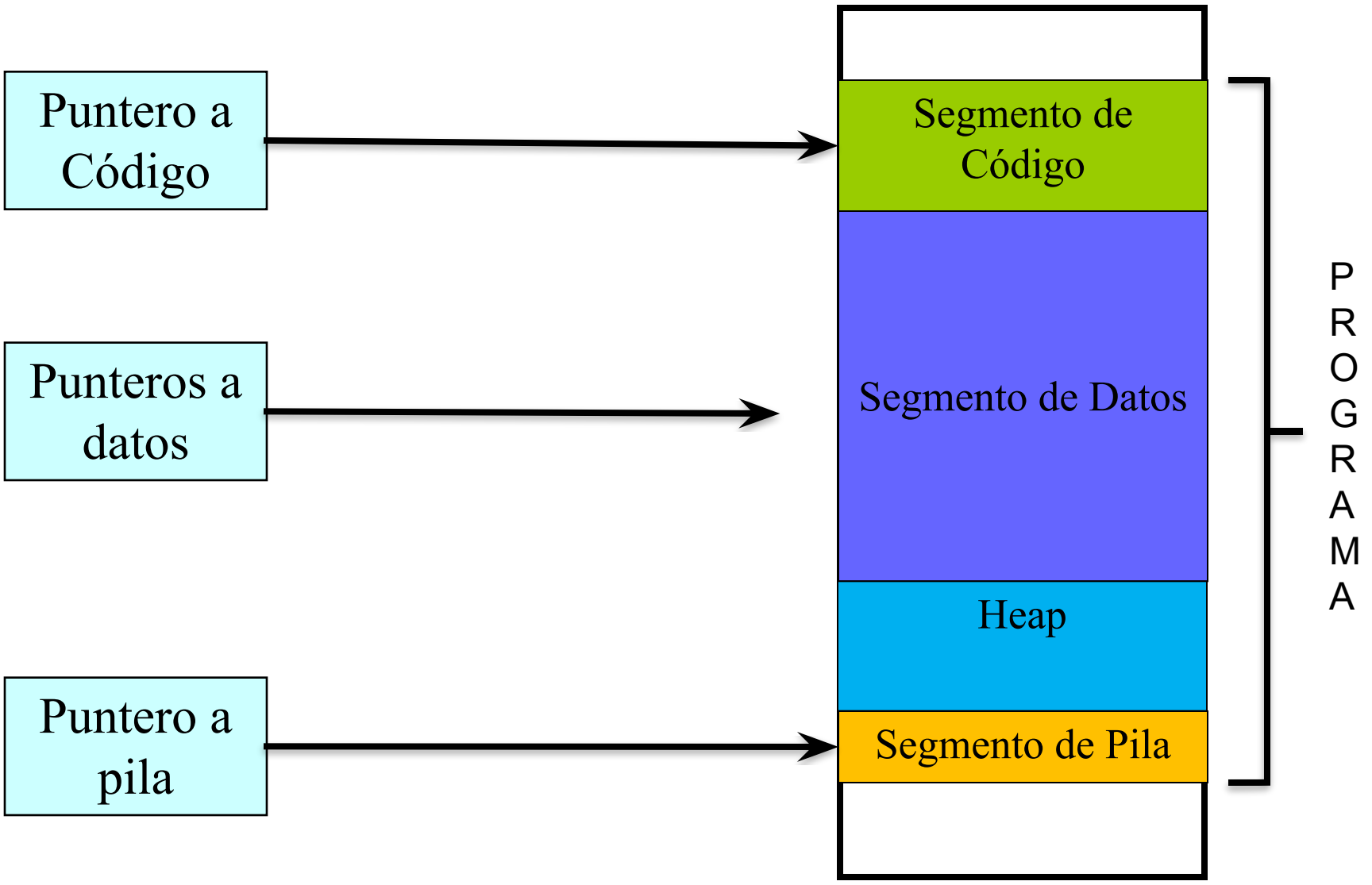


# Programa en memoria

- Código
  - Instrucciones del programa
- Datos
  - Variables estáticas y globales que se inician al cargar el programa.
- Heap
  - Memoria dinámica que se reserva y se libera en tiempo de ejecución.
- Pila
  - Argumentos y variables locales a la función.

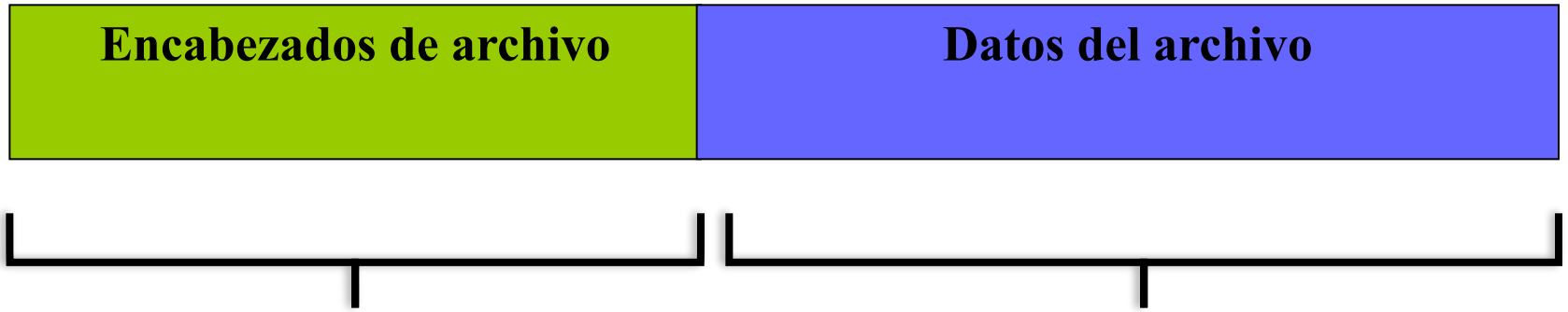


# Programa en memoria (2)



# Programa en disco

Ejemplo archivo **a.out**



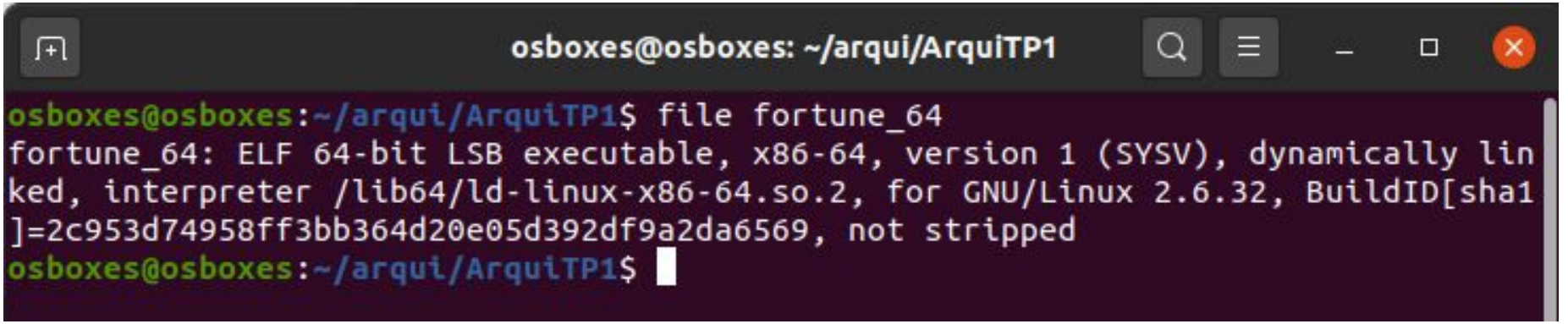

- Encabezados de programa (Segmentos)
- Encabezado de Secciones
  - (Secciones: text, data, rodata, etc)

- Código
- Datos

# Análisis de binario (en Linux)

Archivo ejecutable **fortune_64** (EJ1 de TP1)

Veamos qué tipo de archivo es con el comando **file**




**Vemos que es un ELF (Executable Linux Format) de 64 bits que usa librerías dinámicas**

# Análisis de binario (en Linux)

Lo corremos ……

```
osboxes@osboxes:~/arqui/ArquiTP1$ ./fortune_64
Bienvenido a al adibinador de la fortuna! Este mensaje es muy largo, y puede ser
 muy molesto al usuario. Tal vez deberiamos acortarlo?
Cual es tu nonbre?: Santiago

Tu fortuna es:
 You will forget that you ever knew me.
osboxes@osboxes:~/arqui/ArquiTP1$
```

**Vemos que básicamente pide un nombre e informa un texto tipo galleta de la fortuna.**

# Análisis de binario (en Linux)

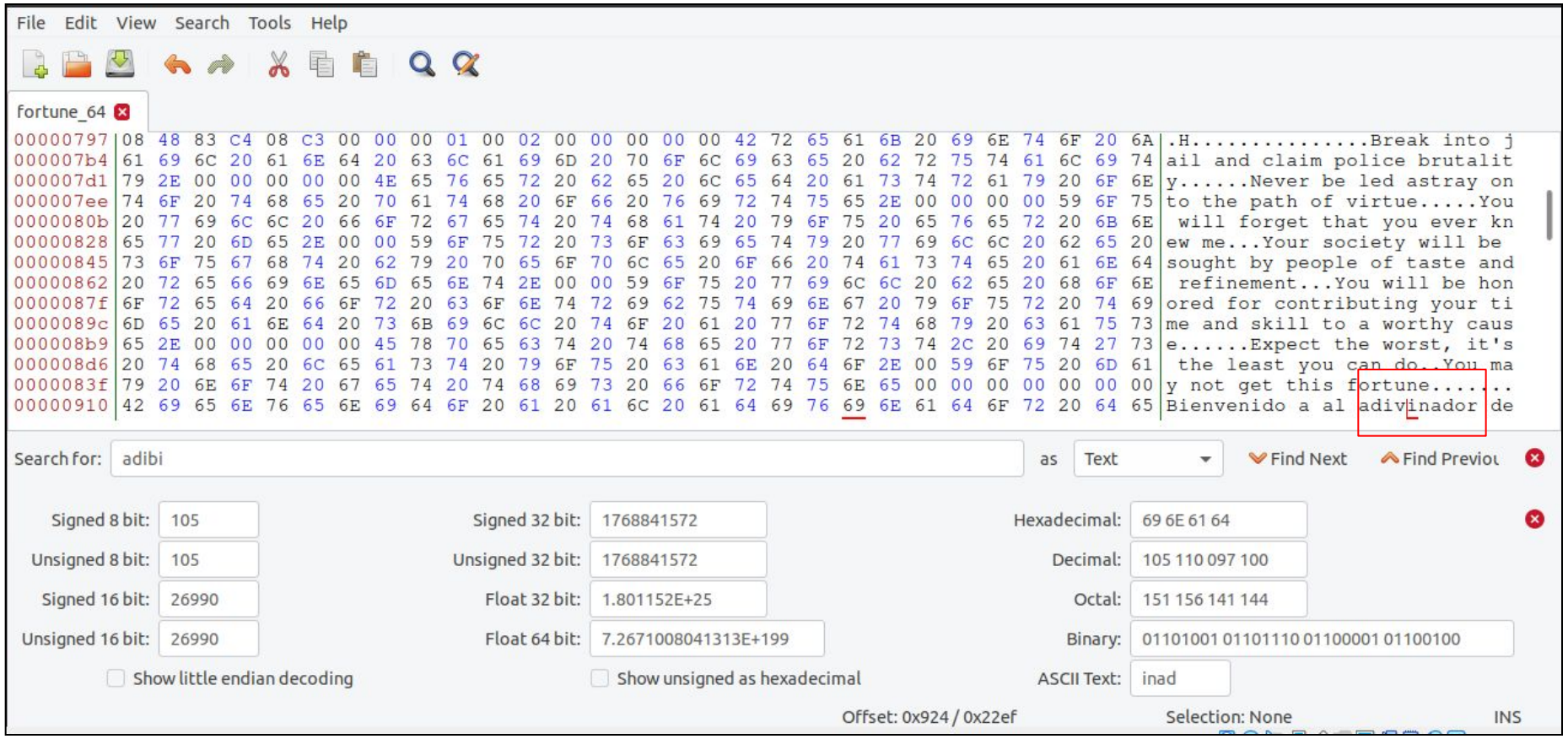
Veamos si podemos extraer todas las cadenas de caracteres con el programa **strings**

```
osboxes@osboxes:~/arqui/ArquiTP1$ strings fortune_64
/lib64/ld-linux-x86-64.so.2
libc.so.6
__isoc99_scanf
puts
__stack_chk_fail
printf
__libc_start_main
__gmon_start__
GLIBC_2.7
GLIBC_2.4
GLIBC_2.2.5
AWAVA
AUATL
[]A\A]A^A_
Break into jail and claim police brutality.
Never be led astray onto the path of virtue.
You will forget that you ever knew me.
Your society will be sought by people of taste and refinement.
You will be honored for contributing your time and skill to a worthy cause.
Expect the worst, it's the least you can do.
You may not get this fortune
Bienvenido a al adibinador de la fortuna! Este mensaje es muy largo, y puede ser
 muy molesto al usuario. Tal vez deberiamos acortarlo?
Cual es tu nonbre?:
Tu fortuna es:
;*3$"
```

**Comprobamos que las cadenas de texto se guardan en texto plano en el archivo (que luego será el segmento de datos del programa)**

# Análisis de binario (en Linux)

Podremos cambiar un texto y alterar el ejecutable ?
Corramos el editor hexadecimal **bless** y arreglemos un error de ortografía




**Guardamos el ejecutable.**

# Análisis de binario (en Linux)

Volvemos a correr el programa

```
osboxes@osboxes:~/arqui/ArquiTP1$ ./fortune_64
Bienvenido a al adivinador de la fortuna! Este mensaje es muy lar
go, y puede ser muy molesto al usuario. Tal vez deberiamos acorta
rlo?
Cual es tu nonbre?:
```

**Pudimos cambiar un carácter.**

**¿Qué conclusiones sacamos?**

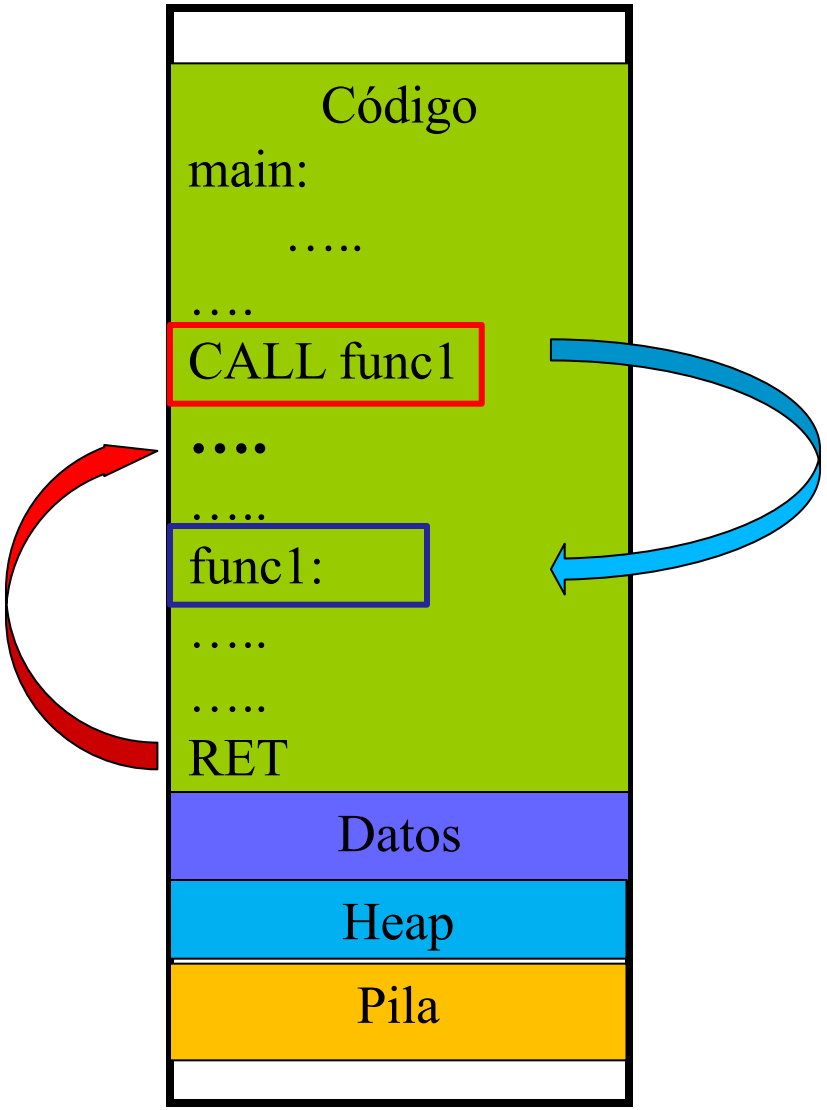
**¿Qué otras cosas podríamos cambiar ?**

**¿Podremos cambiar código alterando con un editor ?**

# Llamadas a Funciones (call)

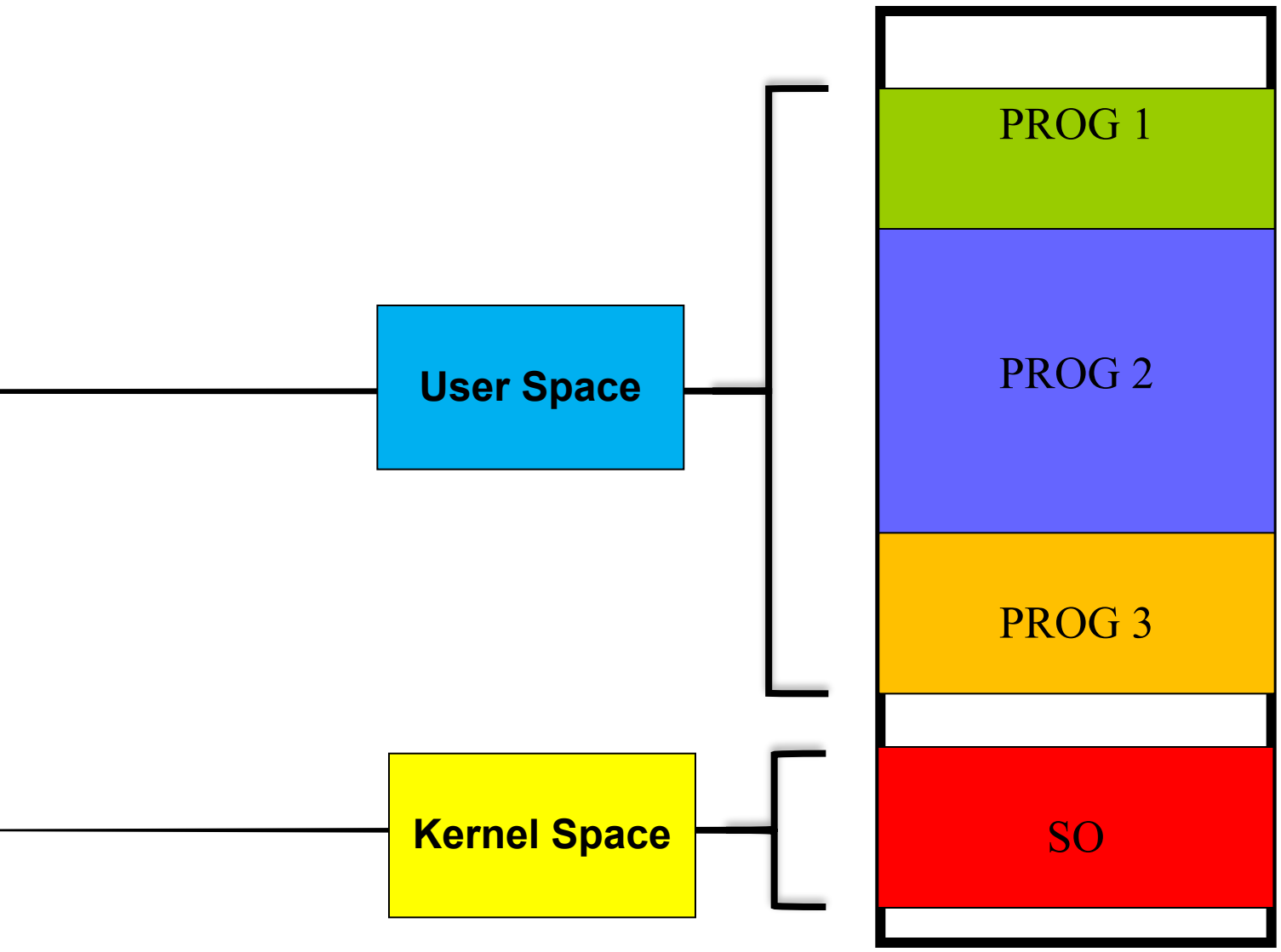
# Programa en memoria

- CALL
  - Instrucción de ASM que permite el llamado a una función.
  - Guarda en la pila la dirección de retorno.
  - La función debe terminar con la instrucción RET de ASM

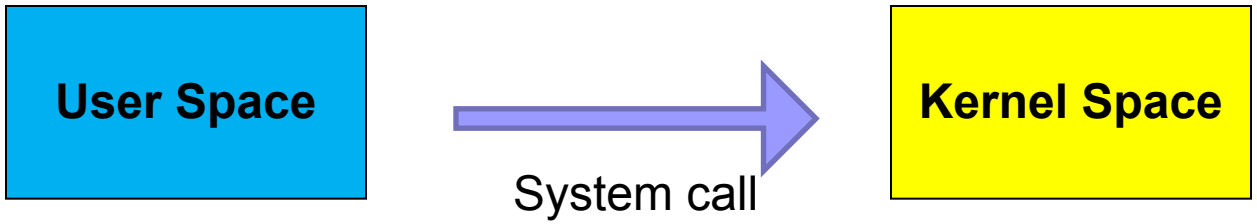


# Llamadas a Sistema Operativo (system calls)

# Introducción a SO



# System Calls




Formas de ejecutar system call

- INT 80h
  - Interrupción nro 80. Consume más tiempo
- Instrucciones **SYSCALL**/SYSRET (Intel) y SYSENTER/SYSEXIT (AMD)
  - Disponibles desde Pentium II
  - La librería de C depende de la arquitectura
- vsyscall y VDSO
  - Syscall virtuales y Virtual Dynamic Shared Object
  - Linux crea páginas de memoria en user space para acelerar tiempos

# Ejemplo de syscall

- Vemos los syscall que utiliza el programa Hola Mundo en C
- "$ strace ./holamundo"

```
svalles@pampero:~/arq$ strace ./holamundo
execve("./holamundo", ["./holamundo"], [/* 16 vars */]) = 0
brk(0)                                  = 0x9765000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7733000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=43016, ...}) = 0
mmap2(NULL, 43016, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7728000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000\226\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1730024, ...}) = 0
mmap2(NULL, 1743580, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb757e000
mprotect(0xb7721000, 4096, PROT_NONE)   = 0
```

# Ejemplo de syscall

```
mmap2(0xb7722000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a3) = 0xb7722000
mmap2(0xb7725000, 10972, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7725000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb757d000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb757d900, limit:1048575, seg_32bit:1,
contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb7722000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb7756000, 4096, PROT_READ)   = 0
munmap(0xb7728000, 43016)               = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 10), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7732000
write(1, "Hola Mundo", 10Hola Mundo)            = 10
exit_group(0)                           = ?
```

# System Calls en Linux (32 bits)

Linux tiene más de 300 system calls

| %eax | Name | Source | %ebx | %ecx | %edx | %esx | %edi |
|---|---|---|---|---|---|---|---|
| 1 | sys_exit | kernel/exit.c | int | - | - | - | - |
| 2 | sys_fork | arch/i386/kernel/process.c | struct pt_regs | - | - | - | - |
| 3 | sys_read | fs/read_write.c | unsigned int | char * | size_t | - | - |
| 4 | sys_write | fs/read_write.c | unsigned int | const char * | size_t | - | - |
| 5 | sys_open | fs/open.c | const char * | int | int | - | - |
| 6 | sys_close | fs/open.c | unsigned int | - | - | - | - |
| 7 | sys_waitpid | kernel/exit.c | pid_t | unsigned int * | int | - | - |
| 8 | sys_creat | fs/open.c | const char * | int | - | - | - |

# Procesadores y Lenguaje ASM (en Intel)

# Registros de Intel para programas

**IP: Instruction Pointer (EIP en 32 bits y RIP en 64 bits)**

Puntero a las próxima instrucción a ejecutarse.

**SP: Stack Pointer (ESP en 32 bits y RSP en 64 bits)**

Puntero a la pila para guardar y extraer datos.

**Registros de Manejo de Memoria:**

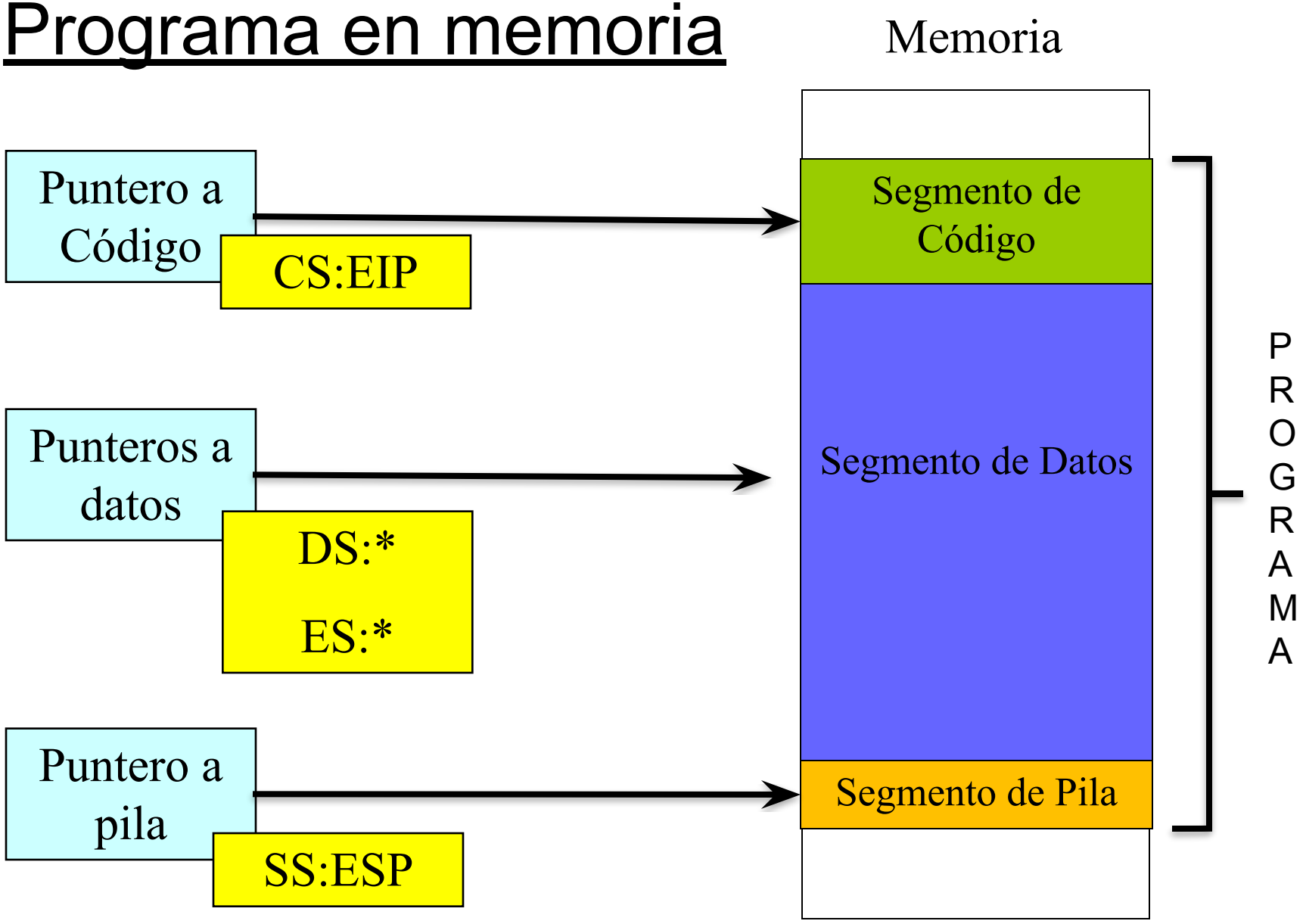
CS: Code Segment.

DS: Data Segment. (también ES, FS y GS)

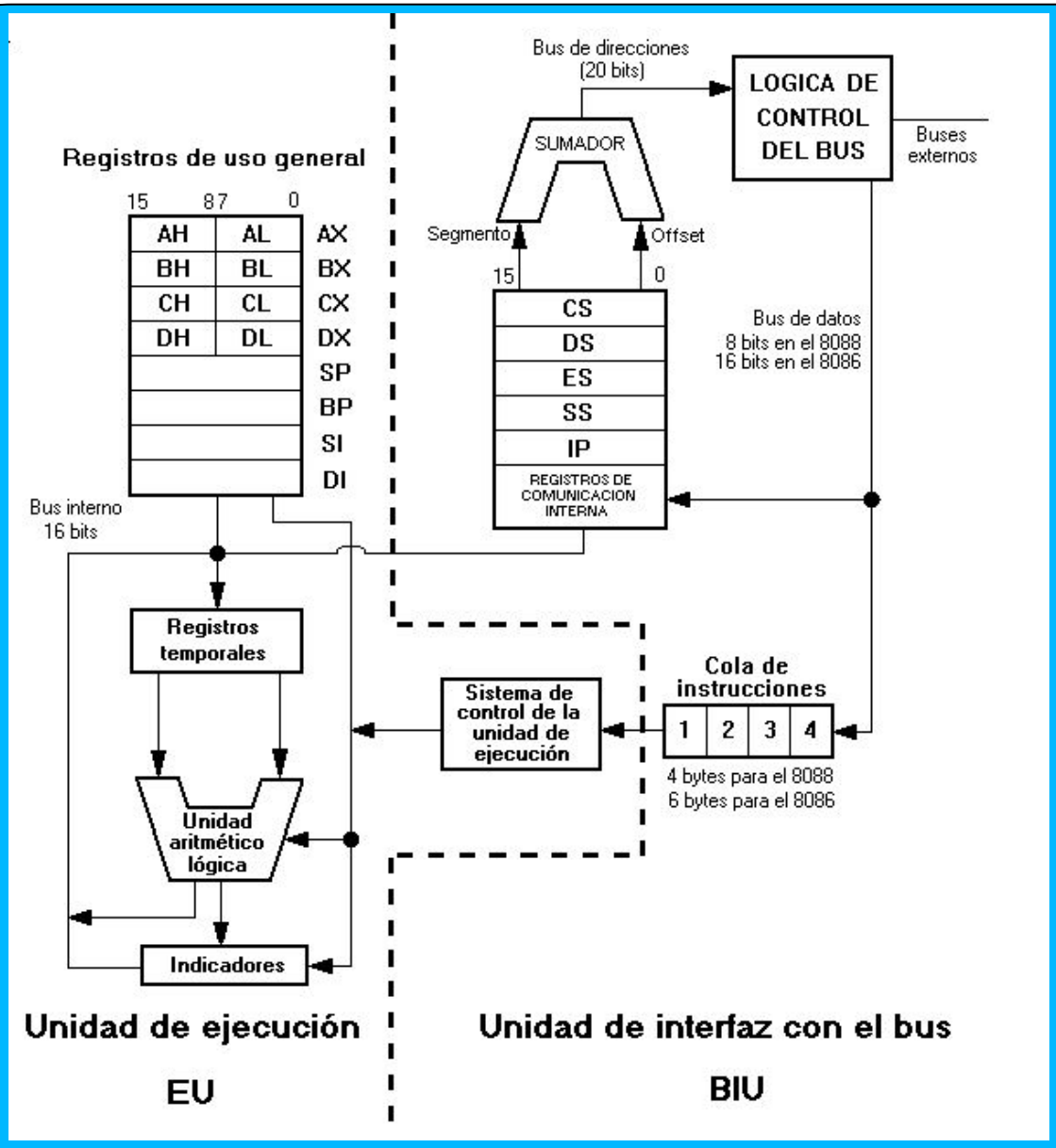
SS: Stack Segment.

(Los registros de segmentos mantienen su tamaño en arquitectura de 32 y 64 bits)

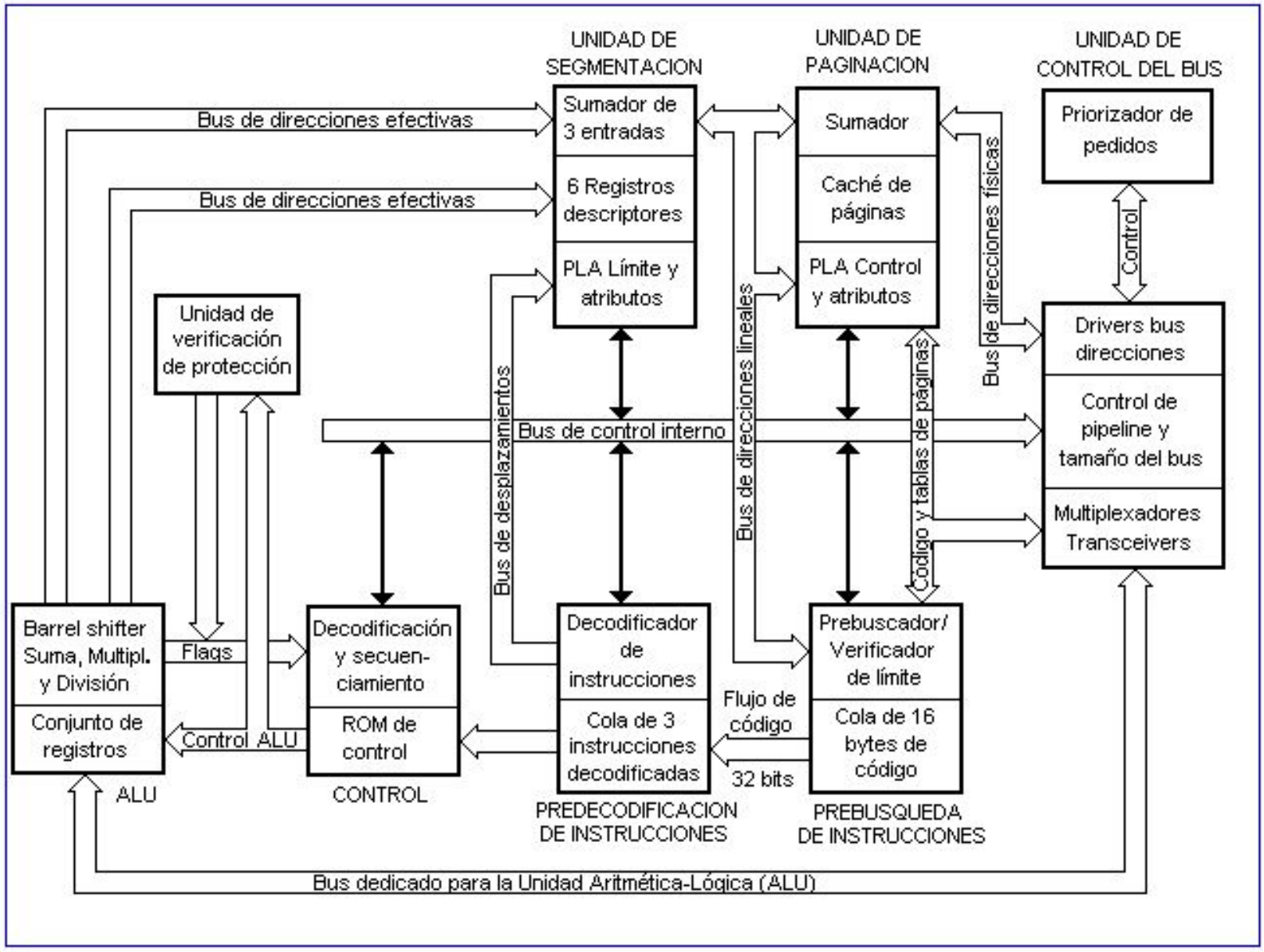
# Programa en memoria



# Arquitectura de 8088/8086 (16 bits)



# Arquitectura de 80386

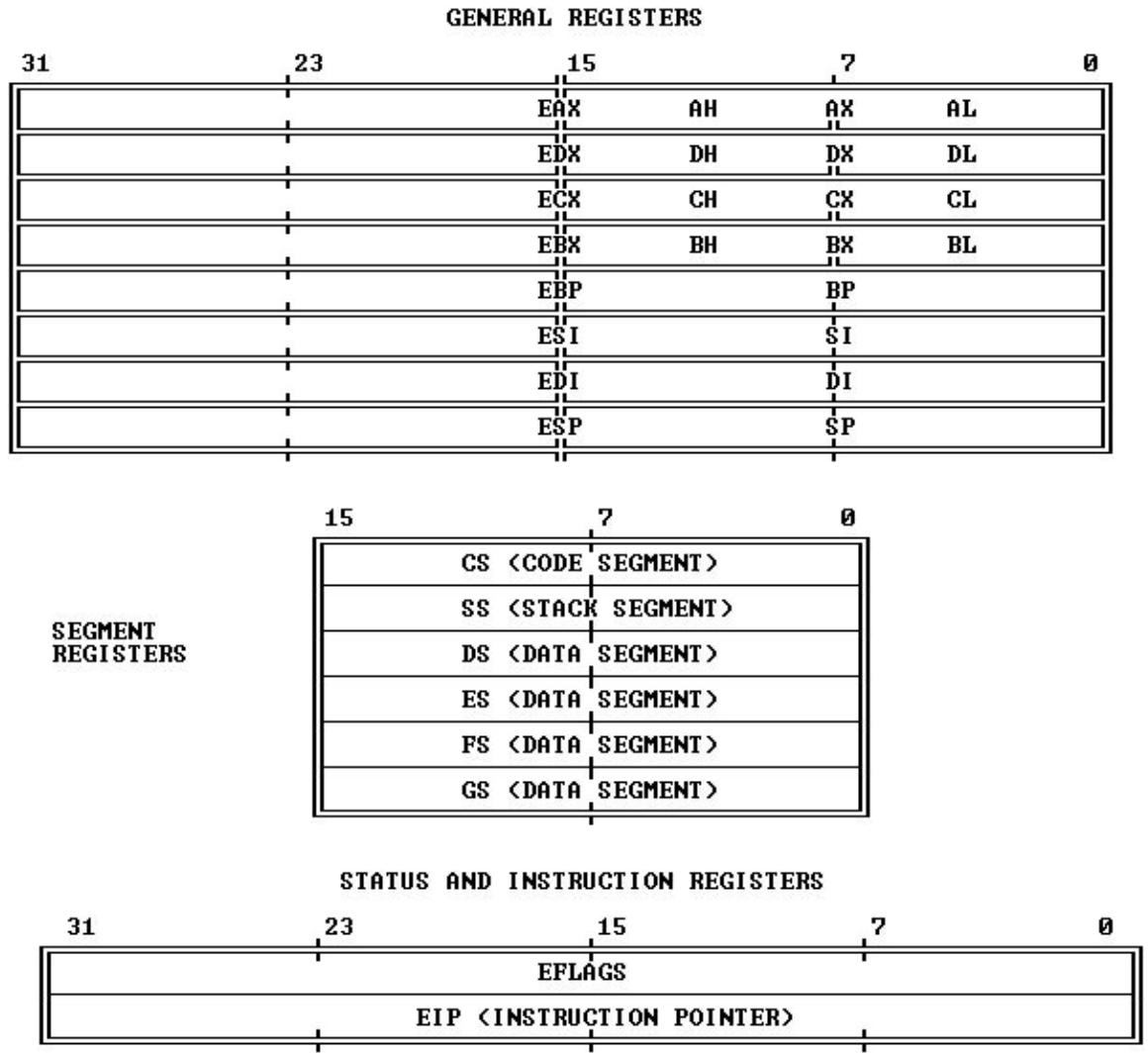


# Arquitectura de 80386

Los procesadores mas avanzados de hoy en día mantienen su arquitectura derivada del 80386.
Se aplica en estos procesadores el “Concepto de diseño superescalar.”, que consiste en agregar mas de una unidad de procesamiento para aumentar la capacidad del mismo.
Cumple con:
**Multitarea**: Existe un requisito importante para los sistemas operativos Multitarea que es tener espacio de memoria individual para cada tarea, y un espacio de memoria común para varias tareas
**Multiusuario**: Que mas de un usuario tenga acceso a la CPU, lo que genera mas tareas.
**Tiempo compartido**: El S.O asigna un tiempo para cada tarea (time slot).
**Tiempo Real**: La conmutación de tareas viene dada por acontecimientos externos.
**Sistema de protección**: Mínimo dos niveles, de Usuario y de Supervisor.
Memoria virtual. Las memorias RAM o ROM siguen siendo caras si se las compara con los precios de los discos rígidos.

# Registros de 80386 (32 bits)

Figure 2-5. 80386 Applications Register Set



# Registros de 64 bits