

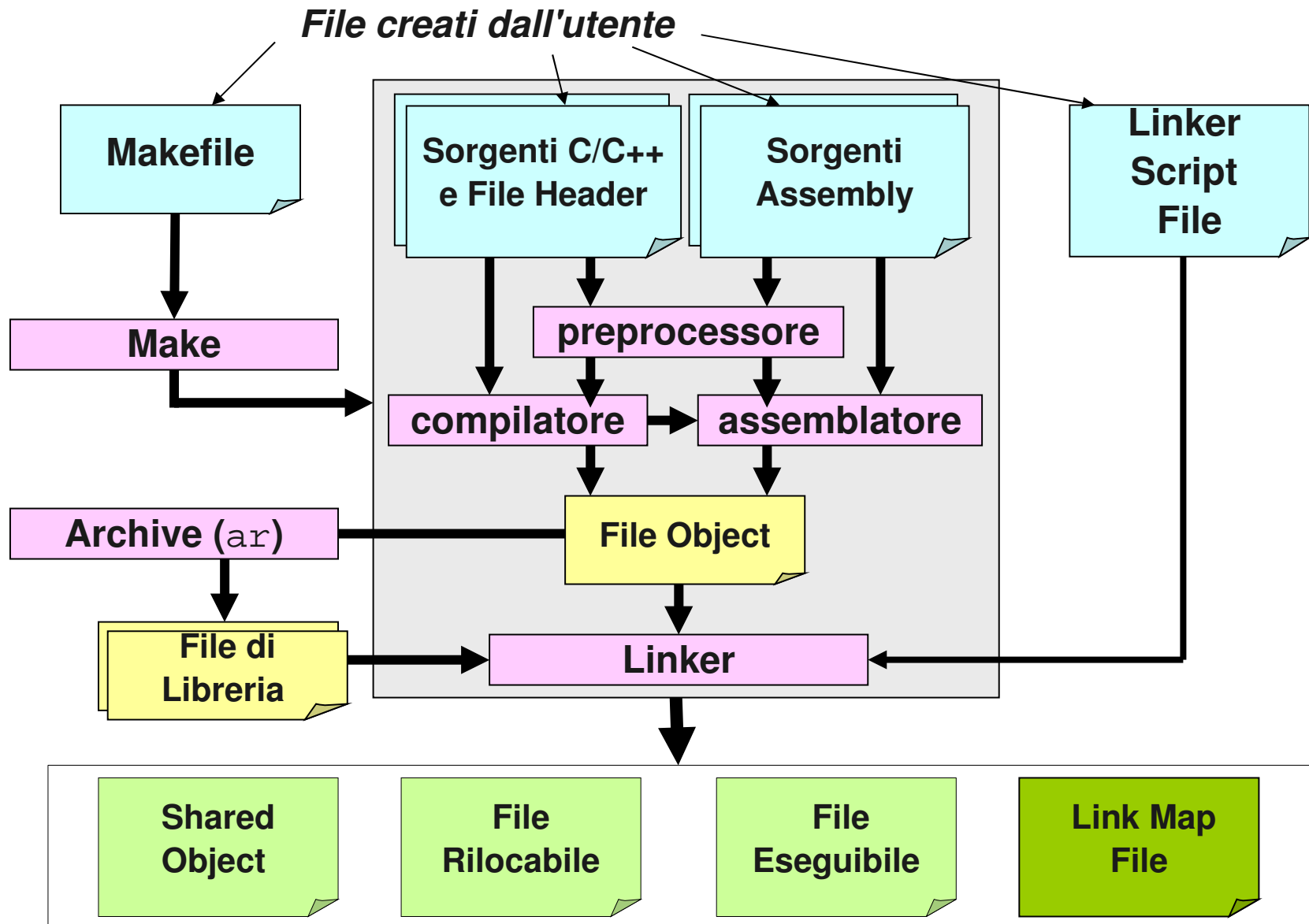
Sistemi Operativi II – Laurea Magistrale in  
Ingegneria Informatica  
Sapienza Università di Roma

Seminario didattico a cura di  
*Alessandro Pellegrini*

**Contenuti:**

1. Formato ELF e strumenti avanzati di compilazione
2. Esempi di strumentazione del codice

# Processo di Compilazione



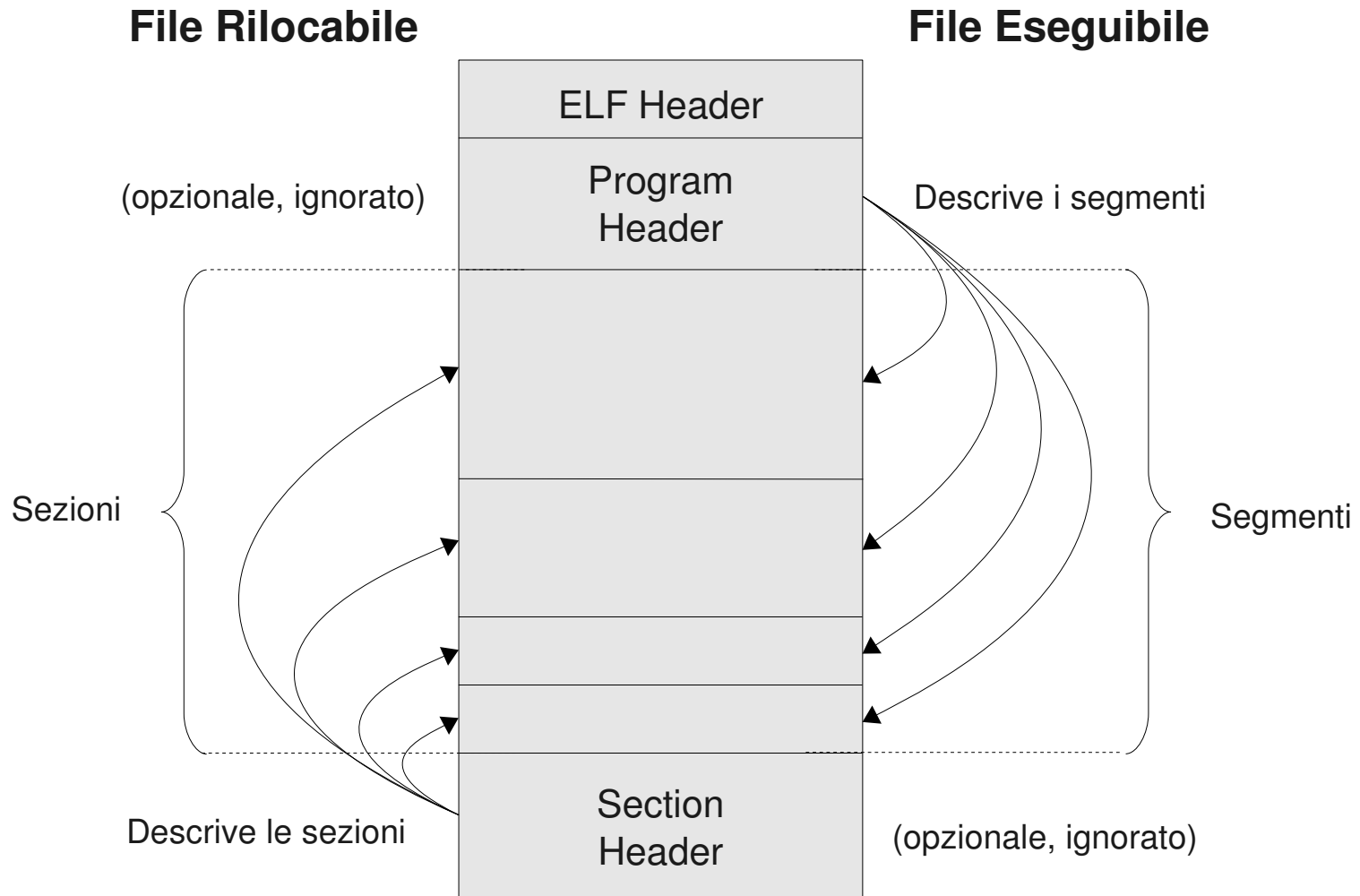
# Formato dei File Object

- Il formato degli eseguibili di \*nix è stato `a.out` per oltre 10 anni.
- I limiti del formato `a.out` erano rappresentati da:
  - cross-compilazione;
  - linking dinamico;
  - creazione semplice di shared library;
  - supporto di inizializzatori/finalizzatori (es: costruttori e distruttori del C++).
- In Linux `a.out` è stato sostituito definitivamente dal formato ELF (*Executable and Linkable Format*) nella versione 1.2 (all'incirca nel 1995).

# Tipologie di File ELF

- ELF definisce il formato dei file binari eseguibili. Ci sono quattro differenti categorie:
  - **Rilocabile** (Creato da compilatori e assembleri. Deve essere processato dal linker prima di poter essere eseguito).
  - **Eseguibile** (Tutti i simboli sono stati risolti eccettuando i simboli delle shared library che devono essere risolti a tempo d'esecuzione).
  - **Shared object** (Libreria condivisa che contiene informazioni sui simboli per il linker e codice direttamente eseguibile a run time).
  - **Core file** (un core dump).
- I file ELF hanno una duplice natura:
  - Compilatori, assembleri e linker trattano i file come un insieme di **sezioni logiche**;
  - Il caricatore di sistema tratta i file come un insieme di **segmenti**.

# Struttura degli ELF



# ELF Header

```
#define EI_NIDENT (16)

typedef struct {
    unsigned char e_ident[EI_NIDENT]; /* Magic number and other info */
    Elf32_Half    e_type;             /* Object file type */
    Elf32_Half    e_machine;         /* Architecture */
    Elf32_Word    e_version;         /* Object file version */
    Elf32_Addr     e_entry;           /* Entry point virtual address */
    Elf32_Off     e_phoff;          /* Program header table file offset */
    Elf32_Off     e_shoff;          /* Section header table file offset */
    Elf32_Word    e_flags;           /* Processor-specific flags */
    Elf32_Half    e_ehsize;          /* ELF header size in bytes */
    Elf32_Half    e_phentsize;     /* Program header table entry size */
    Elf32_Half    e_phnum;         /* Program header table entry count */
    Elf32_Half    e_shentsize;     /* Section header table entry size */
    Elf32_Half    e_shnum;         /* Section header table entry count */
    Elf32_Half    e_shstrndx;     /* Section header string table index */
} Elf32_Ehdr;
```

# File Rilocabili

- Un file **rilocabile** o uno **shared object** è una collezione di sezioni.
- Ciascuna sezione contiene un'unica tipologia di informazioni, come ad esempio codice eseguibile, dati in sola lettura, dati in lettura/scrittura, entry di rilocazione o simboli.
- L'indirizzo di ciascun simbolo viene definito relativamente alla sezione che lo contiene.
  - Pertanto, ad esempio, l'entry point di una funzione è relativo alla sezione del programma che lo contiene.

# Section Header

```
typedef struct {
    Elf32_Word    sh_name;        /* Section name (string tbl index) */
    Elf32_Word    sh_type;        /* Section type */
    Elf32_Word    sh_flags;       /* Section flags */
    Elf32_Addr    sh_addr;        /* Section virtual addr at execution */
    Elf32_Off     sh_offset;      /* Section file offset */
    Elf32_Word    sh_size;        /* Section size in bytes */
    Elf32_Word    sh_link;        /* Link to another section */
    Elf32_Word    sh_info;        /* Additional section information */
    Elf32_Word    sh_addralign;   /* Section alignment */
    Elf32_Word    sh_entsize;    /* Entry size if section holds table */
} Elf32_Shdr;
```



# Tipi e Flag nel Section Header

**PROGBITS:** La sezione racchiude il contenuto del programma (codice, dati, informazioni di debug).

**NOBITS:** Identico a PROGBITS, ma di dimensione nulla.

**SYMTAB** e **DYNSYM:** La sezione contiene tabelle di simboli.

**STRTAB:** La sezione contiene una tabella di stringhe.

**REL** e **RELA:** La sezione contiene informazioni di rilocazione.

**DYNAMIC** e **HASH:** La sezione contiene informazioni relative al linking dinamico.

**WRITE:** La sezione contiene dati scrivibili a tempo d'esecuzione.

**ALLOC:** la sezione occupa memoria durante l'esecuzione del processo.

**EXECINSTR:** La sezione contiene istruzioni macchina eseguibili.

# Alcune Sezioni

- *.text*: contiene le istruzioni del programma
  - › Type: PROGBITS
  - › Flags: ALLOC + EXECINSTR
- *.data*: contiene dati in lettura/scrittura preinizializzati
  - › Type: PROGBITS
  - › Flags: ALLOC + WRITE
- *.rodata*: contiene dati preinizializzati in sola lettura
  - › Type: PROGBITS
  - › Flags: ALLOC
- *.bss*: Contiene dati non inizializzati. Il sistema li imposterà a zero all'avvio dell'esecuzione del programma
  - › Type: NOBITS
  - › Flags: ALLOC + WRITE

# Tabella delle Stringhe

- Le sezioni con tabelle delle stringhe contengono sequenze di caratteri concluse dal terminatore di stringa ' \0 ' .
- I file object utilizzano questa sezione per rappresentare i nomi dei simboli e delle sezioni.
- Viene utilizzato un indice all'interno della tabella per riferire una stringa.
- I nomi dei simboli e la tabella dei simboli sono separati poiché non vi è limite alla lunghezza dei nomi in C/C++

Index	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	\0	n	a	m	e	.	\0	v	a	r
10	i	a	b	l	e	\0	a	b	l	e
20	\0	\0	x	x	\0					

Index	String
0	<i>none</i>
1	<i>name.</i>
7	<i>Variable</i>
11	<i>able</i>
16	<i>able</i>
24	<i>null string</i>

# Tabella dei Simboli

- La tabella dei simboli di un object file mantiene le informazioni necessarie per individuare e rilocare le definizioni simboliche di un programma ed i suoi riferimenti.

```
typedef struct {
    Elf32_Word    st_name;    /* Symbol name */
    Elf32_Addr    st_value;   /* Symbol value */
    Elf32_Word    st_size;    /* Symbol size */
    unsigned char st_info;    /* Symbol binding */
    unsigned char st_other;   /* Symbol visibility */
    Elf32_Section st_shndx;   /* Section index */
} Elf32_Sym;
```

# Tabella di Rilocalizzazione Statica

- La rilocalizzazione è il processo che connette riferimenti a simboli con definizioni di simboli.
- I file rilocabili devono avere informazioni che descrivono come modificare i contenuti delle sezioni.

```
typedef struct {  
    Elf32_Addr    r_offset; /* Address */  
    Elf32_Word    r_info;   /* Relocation type and symbol index */  
} Elf32_Rel;
```

```
typedef struct {  
    Elf32_Addr    r_offset; /* Address */  
    Elf32_Word    r_info;   /* Relocation type and symbol index */  
    Elf32_Sword   r_addend; /* Addend */  
} Elf32_Rela;
```

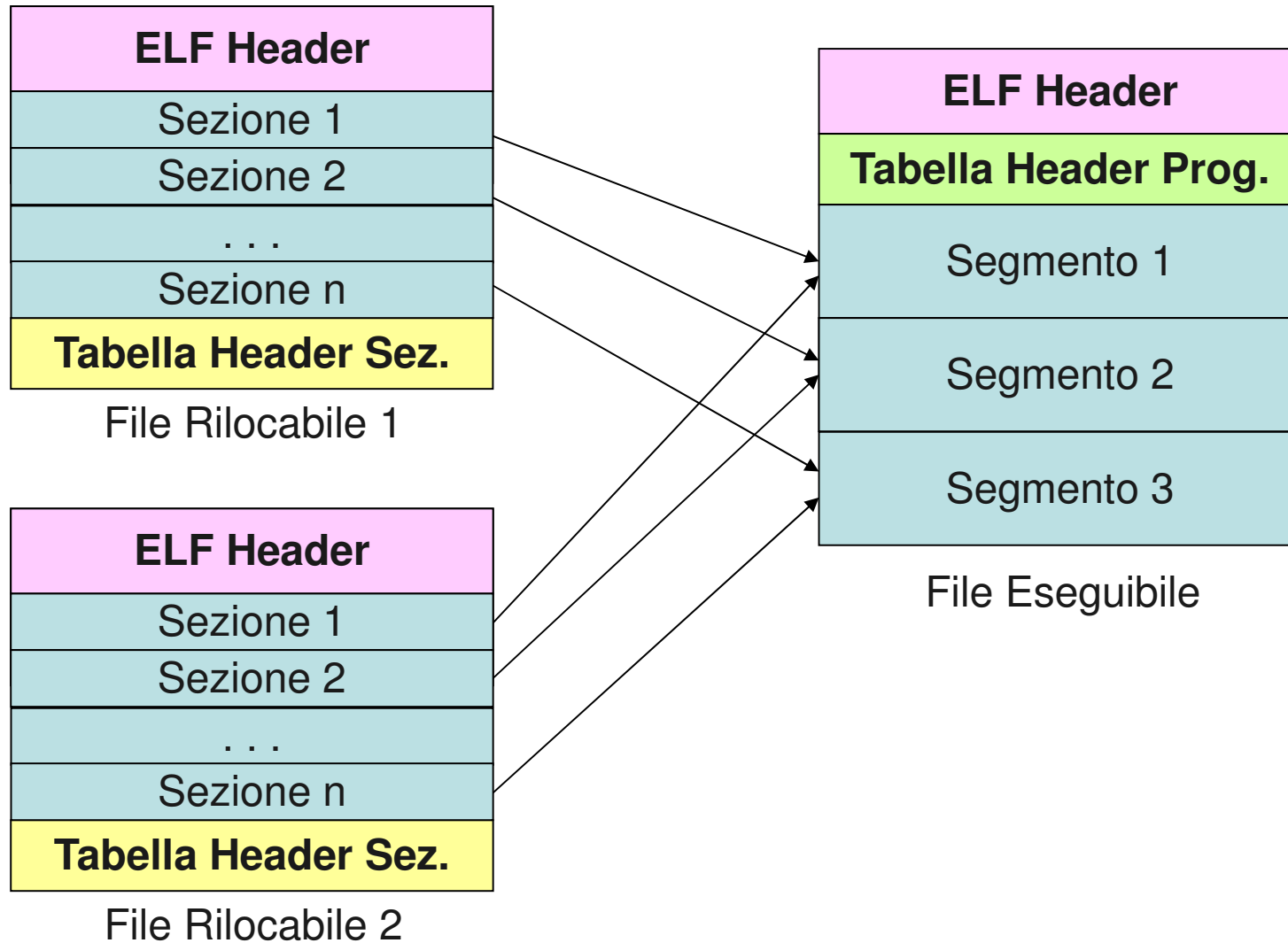
# File Eseguibili

- Di solito un file eseguibile ha soltanto pochi segmenti:
  - Un segmento in sola lettura per il codice.
  - Un segmento in sola lettura per i dati in sola lettura.
  - Un segmento in lettura/scrittura per i dati in lettura/scrittura.
- Tutte le sezioni marcate con il flag `ALLOCATE` vengono impacchettate nei segmenti appropriati, così che il sistema possa mappare il file in memoria con poche operazioni.
  - Ad esempio, se sono presenti le sezioni `.data` e `.bss`, queste verranno inserite tutte all'interno dello stesso segmento in lettura/scrittura.

# Program Header

```
typedef struct {
    Elf32_Word    p_type;    /* Segment type */
    Elf32_Off    p_offset; /* Segment file offset */
    Elf32_Addr  p_vaddr; /* Segment virtual address */
    Elf32_Addr    p_paddr;   /* Segment physical address */
    Elf32_Word  p_filesz; /* Segment size in file */
    Elf32_Word  p_memsz; /* Segment size in memory */
    Elf32_Word  p_flags; /* Segment flags */
    Elf32_Word    p_align;   /* Segment alignment */
} Elf32_Phdr;
```

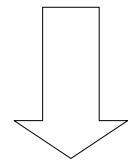
# Ruolo del Linker





# Rilocazione Statica

```
1bc1: e8 fc ff ff ff      call    1bc2 <SystemInit+0x17fe>
1bc6: 83 c4 10             add     $0x10,%esp
1bc9: a1 00 00 00 00      mov     0x0,%eax
```

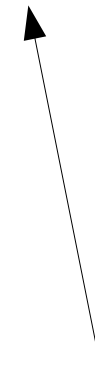
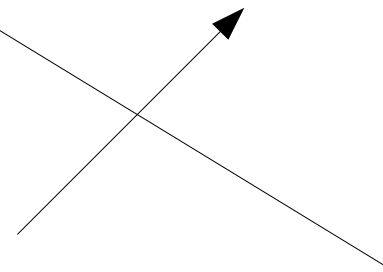
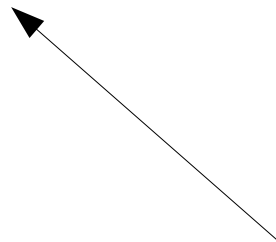
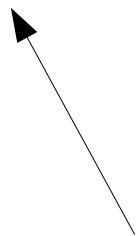


```
8054e59: e8 9a 55 00 00      call    805a3f8 <LogState>
8054e5e: 83 c4 10             add     $0x10,%esp
8054e61: a1 f8 02 06 08      mov     0x80602f8,%eax
```

Posizione delle  
istruzioni

Indirizzi delle  
variabili

Entry point  
delle funzioni



# Direttive al Linker: Linker Script

- La forma più semplice di Linker Script contiene unicamente la direttiva `SECTIONS`;
- Una direttiva `SECTIONS` descrive il layout della memoria del file generato dal linker.

```
SECTIONS
```

```
{
```

```
  . = 0x10000;
```

```
  .text : { *(.text) }
```

```
  . = 0x8000000;
```

```
  .data : { *(.data) }
```

```
  .bss : { *(.bss) }
```

```
}
```

Imposta il valore del *location counter*

Inserisce tutte le sezioni *.text* dei file di input nella sezione *.text* del file di output all'indirizzo specificato dal *location counter*.

# Esempio: codice C

```
#include <stdio.h>
```

```
int xx, yy;
```

```
int main(void) {
```

```
    xx = 1;
```

```
    yy = 2;
```

```
    printf ("xx %d yy %d\n", xx, yy);
```

```
}
```

# Esempio: ELF Header

```
$ objdump -x esempio-elf
```

```
eempio-elf:          file format elf32-i386  
architecture: i386, flags 0x00000112:  
EXEC_P, HAS_SYMS, D_PAGED  
start address 0x08048310
```

# Esempio: Program Header

```
PHDR off      0x00000034 vaddr 0x08048034 paddr 0x08048034 align 2**2
      filesz 0x00000100 memsz 0x00000100 flags r-x
INTERP off     0x00000134 vaddr 0x08048134 paddr 0x08048134 align 2**0
      filesz 0x00000013 memsz 0x00000013 flags r--
LOAD  off     0x00000000 vaddr 0x08048000 paddr 0x08048000 align 2**12
      filesz 0x000004f4 memsz 0x000004f4 flags r-x
LOAD  off     0x00000f0c vaddr 0x08049f0c paddr 0x08049f0c align 2**12
      filesz 0x00000108 memsz 0x00000118 flags rw-
DYNAMIC off    0x00000f20 vaddr 0x08049f20 paddr 0x08049f20 align 2**2
      filesz 0x000000d0 memsz 0x000000d0 flags rw-
NOTE  off     0x00000148 vaddr 0x08048148 paddr 0x08048148 align 2**2
      filesz 0x00000020 memsz 0x00000020 flags r--
STACK off     0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**2
      filesz 0x00000000 memsz 0x00000000 flags rw-
RELRO off     0x00000f0c vaddr 0x08049f0c paddr 0x08049f0c align 2**0
      filesz 0x000000f4 memsz 0x000000f4 flags r--
```

# Esempio: Dynamic Section

NEEDED	<b>libc.so.6</b>
INIT	0x08048298
FINI	0x080484bc
HASH	0x08048168
STRTAB	0x08048200
SYMTAB	0x080481b0
STRSZ	0x0000004c
SYMENT	0x00000010
DEBUG	0x00000000
PLTGOT	0x08049ff4
PLTRELSZ	0x00000018
PLTREL	0x00000011
JMPREL	0x08048280

Indica la necessità di linkare questa shared library per utilizzare printf()

# Esempio: Header delle Sezioni

Idx	Name	Size	VMA	LMA	File off	Algn
2	.hash	00000028	08048168	08048168	00000168	2**2
		CONTENTS,	ALLOC,	LOAD,	READONLY,	DATA
10	.init	00000030	08048298	08048298	00000298	2**2
		CONTENTS,	ALLOC,	LOAD,	READONLY,	CODE
11	.plt	00000040	080482c8	080482c8	000002c8	2**2
		CONTENTS,	ALLOC,	LOAD,	READONLY,	CODE
12	.text	000001ac	08048310	08048310	00000310	2**4
		CONTENTS,	ALLOC,	LOAD,	READONLY,	CODE
13	.fini	0000001c	080484bc	080484bc	000004bc	2**2
		CONTENTS,	ALLOC,	LOAD,	READONLY,	CODE
14	.rodata	00000015	080484d8	080484d8	000004d8	2**2
		CONTENTS,	ALLOC,	LOAD,	READONLY,	ATA
22	.data	00000008	0804a00c	0804a00c	0000100c	2**2
		CONTENTS,	ALLOC,	LOAD,	DATA	
23	.bss	00000010	0804a014	0804a014	00001014	2**2
		ALLOC				

# Esempio: Tabella dei Simboli

```
...
00000000 l      df *ABS*      00000000      esempio-elf.c
08049f0c l          .ctors      00000000      .hidden __init_array_end
08049f0c l          .ctors      00000000      .hidden __init_array_start
08049f20 l      O .dynamic  00000000      .hidden _DYNAMIC
0804a00c w          .data      00000000      data_start
08048420 g      F .text      00000005      __libc_csu_fini
08048310 g      F .text      00000000      _start
00000000 w          *UND*      00000000      __gmon_start__
...
08049f18 g      O .dtors      00000000      .hidden __DTOR_END__
08048430 g      F .text      0000005a      __libc_csu_init
00000000      F *UND*      00000000      printf@@GLIBC_2.0
0804a01c g      O .bss      00000004      yy
0804a014 g          *ABS*      00000000      __bss_start
0804a024 g          *ABS*      00000000      _end
0804a014 g          *ABS*      00000000      _edata
0804848a g      F .text      00000000      .hidden __i686.get_pc_thunk.bx
080483c4 g      F .text      0000004d      main
08048298 g      F .init      00000000      _init
0804a020 g      O .bss      00000004      xx
```



# Visibilità dei Simboli

- Simboli *weak*:

- Più moduli possono avere lo stesso simbolo *weak*;
- L'entità dichiarata non può essere scavalcata da altri moduli;
- Utile per librerie che non vogliono entrare in conflitto con programmi di utenti.

- gcc versione 4.0 fornisce l'opzione a riga di comando

- fvisibility*:

- `default`: comportamento normale, la dichiarazione è visibile dagli altri moduli;
- `hidden`: due dichiarazioni di un oggetto si riferiscono allo stesso oggetto se sono nello stesso `shared object`;
- `internal`: un'entità dichiarata in un modulo non può essere riferita neppure tramite puntatori;
- `protected`: la dichiarazione verrà trasformata in un simbolo *weak*;

# Visibilità dei Simboli (2)

```
int variable __attribute__((visibility ("hidden")));
```

```
#pragma GCC visibility push(hidden)  
int variable;
```

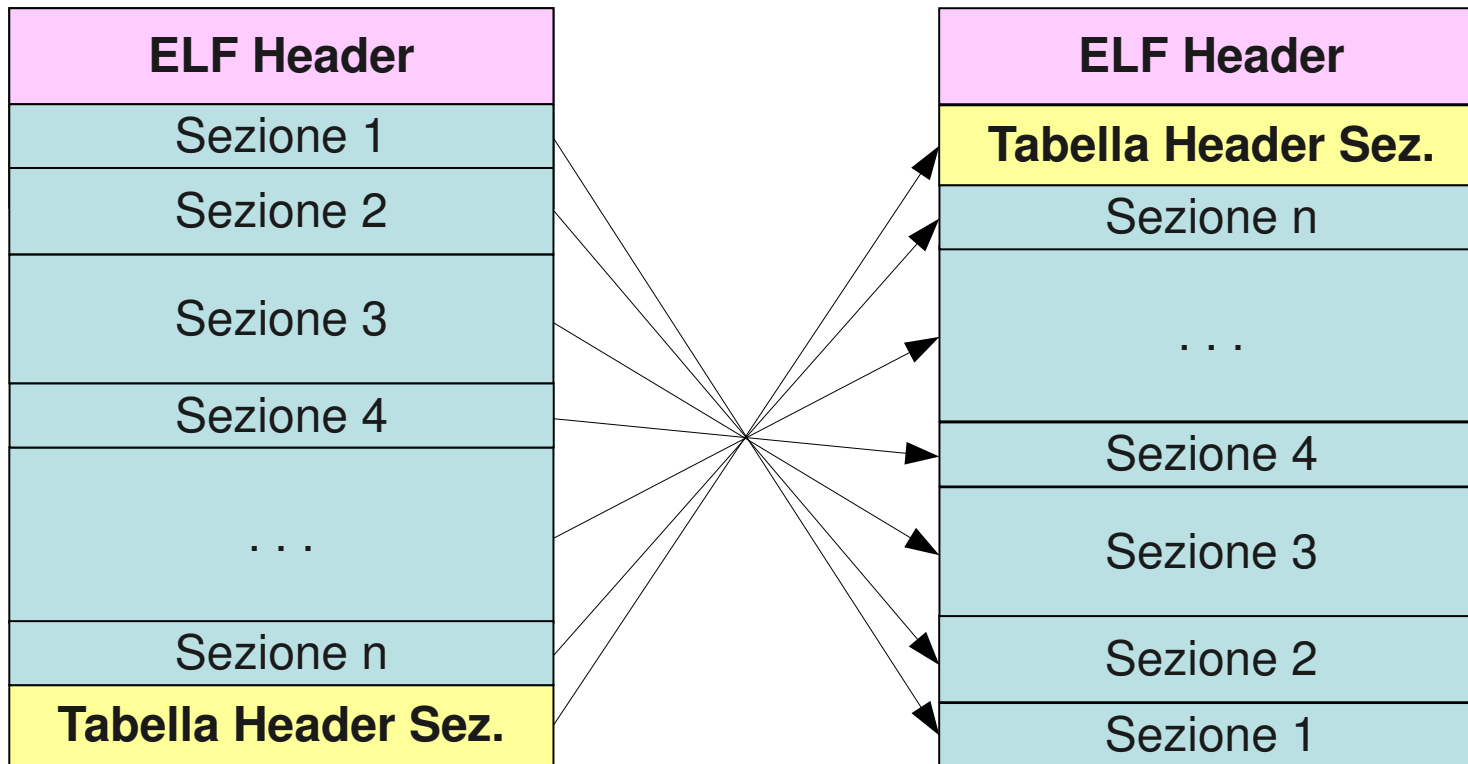
```
int increment(void) {  
    return ++variable;  
}
```

```
#pragma GCC visibility pop
```

# Avvio di un Programma: Linux Loader

- In `fs/exec.c`:
  - **do\_execve()**:
    - Circa 50 righe di codice;
    - Effettua controllo sugli errori e riempie la struttura `struct linux_binprm`;
    - Cerca un handler di file binari.
  - **search\_binary\_handler()**:
    - Scandisce una lista di handler di formati binari registrati;
    - Se nessun handler riconosce l'immagine, la system call restituisce il codice di errore `ENOEXEC` (“Exec Format Error”);
- In `fs/binfmt_elf.c`:
  - **load\_elf\_binary()**:
    - Carica in memoria l'immagine del file tramite `mmap`;
    - Legge l'header ed imposta i permessi.

# Modifica di un File ELF: Riordino



# Modifica di un File ELF: Riordino (2)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <elf.h>
```

Per poter utilizzare le strutture  
che descrivono i file ELF

```
int main(int argc, char **argv) {
```

```
    int elf_src, elf_dst, file_size, i;
    char *src_image, *dst_image, *ptr;
    Elf32_Ehdr *ehdr_src, *ehdr_dst;
    Elf32_Shdr *shdr_src, *shdr_dst;
```

```
    if((elf_src = open(argv[1], O_RDONLY)) == -1) exit(-1);
    if((elf_dst = creat(argv[2], 0644)) == -1) exit(-1);
    file_size = lseek(elf_src, 0L, SEEK_END);
    lseek(elf_src, 0L, SEEK_SET);
    src_image = malloc(file_size);
    ptr = dst_image = malloc(file_size);
    read(elf_src, src_image, file_size);
    ehdr_src = (Elf32_Ehdr *)src_image;
    ehdr_dst = (Elf32_Ehdr *)dst_image;
```

```
    memcpy(ptr, src_image, sizeof(Elf32_Ehdr));
    ptr += sizeof(Elf32_Ehdr);
```

Gli ELF header dei due file sono  
(sostanzialmente) identici

# Modifica di un File ELF: Riordino (3)

```
shdr_dst = (Elf32_Shdr *)ptr;
shdr_src = (Elf32_Shdr *)(src_image + ehdr_src->e_shoff);
ehdr_dst->e_shoff = sizeof(Elf32_Ehdr);
ptr += ehdr_src->e_shnum * ehdr_dst->e_shentsize;
```

Corregge la posizione degli header delle sezioni

```
memcpy(shdr_dst, shdr_src, sizeof(Elf32_Shdr));
```

Copia le sezioni e gli header

```
for(i = ehdr_src->e_shnum - 1; i > 0; i--) {
```

```
    memcpy(shdr_dst + ehdr_src->e_shnum - i, shdr_src + i, sizeof(Elf32_Shdr));
    memcpy(ptr, src_image + shdr_src[i].sh_offset, shdr_src[i].sh_size);
    shdr_dst[ehdr_src->e_shnum - i].sh_offset = ptr - dst_image;
```

```
    if(shdr_src[i].sh_link > 0)
        shdr_dst[ehdr_src->e_shnum - i].sh_link = ehdr_src->e_shnum - shdr_src[i].sh_link;
```

```
    if(shdr_src[i].sh_info > 0)
        shdr_dst[ehdr_src->e_shnum - i].sh_info = ehdr_src->e_shnum - shdr_src[i].sh_info;
```

```
    ptr += shdr_src[i].sh_size;
```

```
}
```

```
ehdr_dst->e_shstrndx = ehdr_src->e_shnum - ehdr_src->e_shstrndx;
```

```
write(elf_dst, dst_image, file_size);
close(elf_src);
close(elf_dst);
```

```
}
```

# Modifica di un File ELF: Riordino (4)

```
$ readelf -S esempio-elf.o
```

```
There are 11 section headers, starting at offset 0x108:
```

```
Section Headers:
```

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[ 0]		NULL	00000000	000000	000000	00		0	0	0
[ 1]	.text	PROGBITS	00000000	000034	00004d	00	AX	0	0	4
[ 2]	.rel.text	REL	00000000	0003a4	000030	08		9	1	4
[ 3]	.data	PROGBITS	00000000	000084	000000	00	WA	0	0	4
[ 4]	.bss	NOBITS	00000000	000084	000000	00	WA	0	0	4
[ 5]	.rodata	PROGBITS	00000000	000084	00000d	00	A	0	0	1
[ 6]	.comment	PROGBITS	00000000	000091	000025	00		0	0	1
[ 7]	.note.GNU-stack	PROGBITS	00000000	0000b6	000000	00		0	0	1
[ 8]	.shstrtab	STRTAB	00000000	0000b6	000051	00		0	0	1
[ 9]	.symtab	SYMTAB	00000000	0002c0	0000c0	10		10	8	4
[10]	.strtab	STRTAB	00000000	000380	000021	00		0	0	1

# Modifica di un File ELF: Riordino (5)

```
$ readelf -S riordinato.o
```

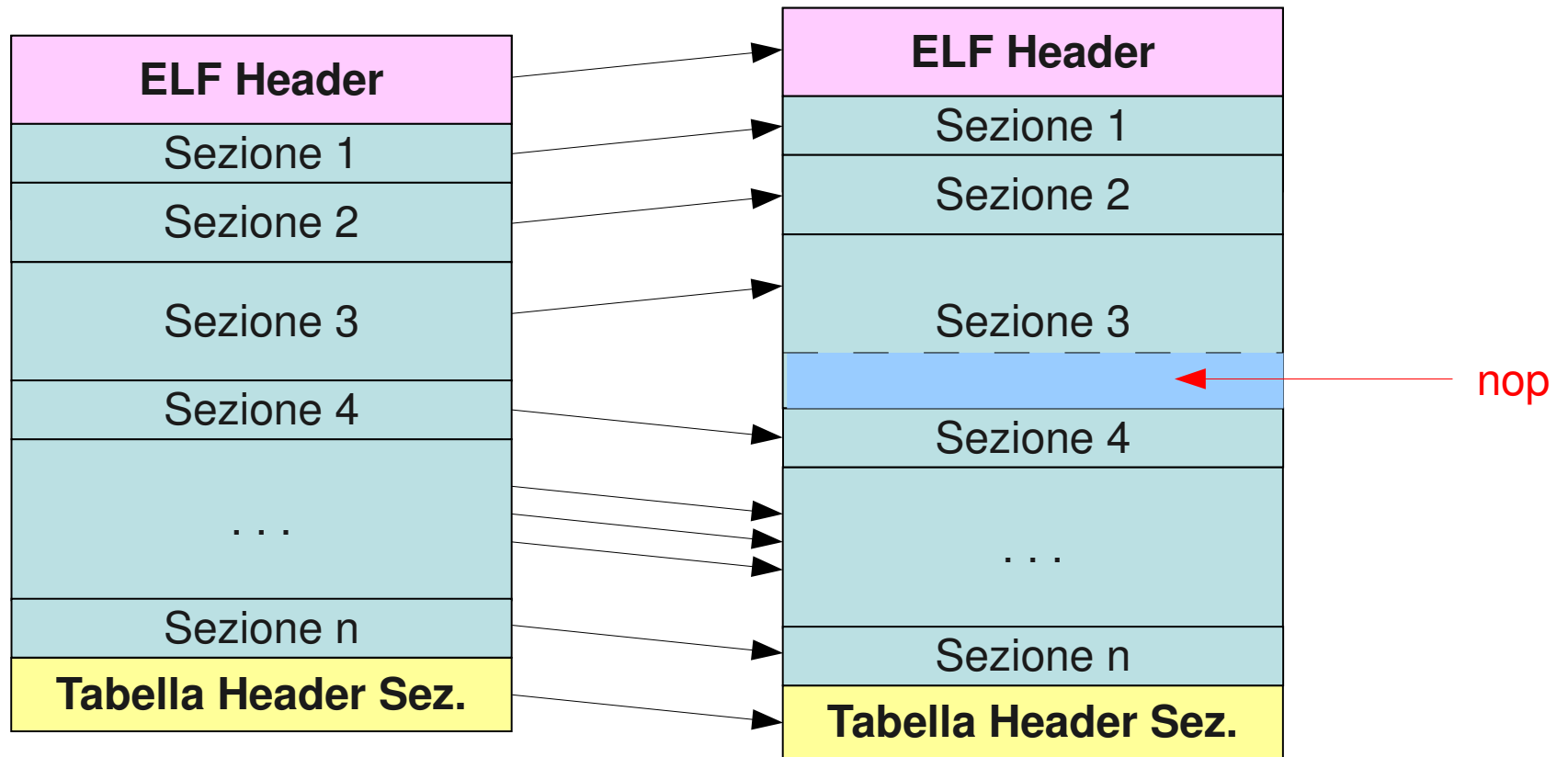
```
There are 11 section headers, starting at offset 0x34:
```

```
Section Headers:
```

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[ 0]		NULL	00000000	000000	000000	00		0	0	0
[ 1]	.strtab	STRTAB	00000000	0001ec	000021	00		0	0	1
[ 2]	.symtab	SYMTAB	00000000	00020d	0000c0	10		1	3	4
[ 3]	.shstrtab	STRTAB	00000000	0002cd	000051	00		0	0	1
[ 4]	.note.GNU-stack	PROGBITS	00000000	00031e	000000	00		0	0	1
[ 5]	.comment	PROGBITS	00000000	00031e	000025	00		0	0	1
[ 6]	.rodata	PROGBITS	00000000	000343	00000d	00	A	0	0	1
[ 7]	.bss	NOBITS	00000000	000350	000000	00	WA	0	0	4
[ 8]	.data	PROGBITS	00000000	000350	000000	00	WA	0	0	4
[ 9]	.rel.text	REL	00000000	000350	000030	08		2	10	4
[10]	.text	PROGBITS	00000000	000380	00004d	00	AX	0	0	4



# Modifica di un File ELF: nop



# Modifica di un File ELF: nop (2)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <elf.h>

#define NOP_NUM 10
#define NOP_CODE 0x90 // 1 byte
#define SEC_NUM 1

int main(int argc, char **argv) {

    int elf_src, elf_dst, file_size, i;
    char *src_image, *dst_image;
    Elf32_Ehdr *ehdr_src;
    Elf32_Shdr *shdr_src, *shdr_dst;

    if((elf_src = open(argv[1], O_RDONLY)) == -1) exit(-1);
    if((elf_dst = creat(argv[2], 0644)) == -1) exit(-1);
    file_size = lseek(elf_src, 0L, SEEK_END);
    lseek(elf_src, 0L, SEEK_SET);
    src_image = malloc(file_size);
    dst_image = malloc(file_size + NOP_NUM);
    read(elf_src, src_image, file_size);

    ehdr_src = (Elf32_Ehdr *)src_image;
    shdr_src = (Elf32_Shdr *)(src_image + ehdr_src->e_shoff);
```

# Modifica di un File ELF: nop (3)

```
shdr_dst = (Elf32_Shdr *) (dst_image + ehdr_src->e_shoff + NOP_NUM);
```

```
memcpy(dst_image, src_image, sizeof(Elf32_Ehdr));
```

```
((Elf32_Ehdr *)dst_image)->e_shoff += NOP_NUM;
```

```
for(i = 0; i <= SEC_NUM; i++)
```

```
    memcpy(dst_image + shdr_src[i].sh_offset, src_image + shdr_src[i].sh_offset,  
           shdr_src[i].sh_size);
```

Inserisce le nop



```
memset(dst_image + shdr_src[SEC_NUM].sh_offset + shdr_src[SEC_NUM].sh_size, NOP_CODE, NOP_NUM);
```

```
for(i = SEC_NUM + 1; i < ehdr_src->e_shnum; i++)
```

```
    memcpy(dst_image + shdr_src[i].sh_offset + NOP_NUM, src_image + shdr_src[i].sh_offset,  
           shdr_src[i].sh_size);
```

```
for(i = 0; i <= SEC_NUM; i++)
```

```
    memcpy(shdr_dst + i, shdr_src + i, sizeof(Elf32_Shdr));
```

```
shdr_dst[SEC_NUM].sh_size += NOP_NUM;
```

Corregge la dimensione  
della sezione



```
for(i = SEC_NUM + 1; i < ehdr_src->e_shnum; i++) {
```

```
    memcpy(shdr_dst + i, shdr_src + i, sizeof(Elf32_Shdr));
```

```
    shdr_dst[i].sh_offset += NOP_NUM;
```



Trasla in avanti le  
altre sezioni

```
write(elf_dst, dst_image, file_size + NOP_NUM);
```

```
close(elf_src);
```

```
close(elf_dst);
```

```
}
```

# Modifica di un File ELF: nop (4)

```
$ objdump -S esempio-elf.o
```

```
Disassembly of section .text:
```

```
00000000 <main>:
```

```
  0:  8d 4c 24 04          lea    0x4(%esp),%ecx
  4:  83 e4 f0            and    $0xffffffff0,%esp
  7:  ff 71 fc           pushl  -0x4(%ecx)
  a:  55                push  %ebp
  b:  89 e5             mov    %esp,%ebp
  d:  51                push  %ecx
  e:  83 ec 14          sub    $0x14,%esp
 11:  c7 05 00 00 00 00 01 movl   $0x1,0x0
 18:  00 00 00
[...]
```

```
 38:  c7 04 24 00 00 00 00 movl   $0x0,(%esp)
 3f:  e8 fc ff ff ff     call  40 <main+0x40>
 44:  83 c4 14           add    $0x14,%esp
 47:  59                pop    %ecx
 48:  5d                pop    %ebp
 49:  8d 61 fc          lea   -0x4(%ecx),%esp
 4c:  c3                ret
```

# Modifica di un File ELF: nop (5)

```
$ objdump -S nop.o
```

```
Disassembly of section .text:
```

```
00000000 <main>:
```

```
  0:  8d 4c 24 04          lea    0x4(%esp),%ecx
  4:  83 e4 f0            and    $0xffffffff0,%esp
  7:  ff 71 fc           pushl  -0x4(%ecx)
  a:  55                 push   %ebp
  b:  89 e5              mov    %esp,%ebp
  d:  51                 push   %ecx
  e:  83 ec 14           sub    $0x14,%esp
 11:  c7 05 00 00 00 00 01 movl   $0x1,0x0
 18:  00 00 00
```

```
[...]
```

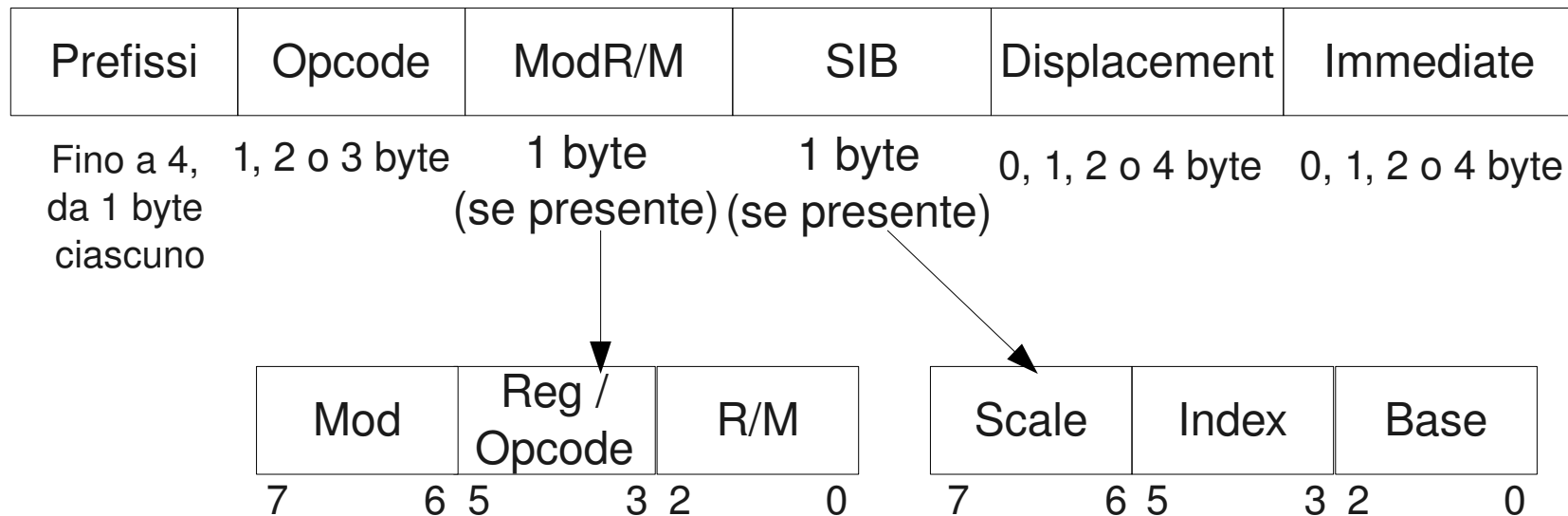
```
 38:  c7 04 24 00 00 00 00 movl   $0x0,(%esp)
 3f:  e8 fc ff ff ff     call   40 <main+0x40>
 44:  83 c4 14           add    $0x14,%esp
 47:  59                 pop    %ecx
 48:  5d                 pop    %ebp
 49:  8d 61 fc           lea   -0x4(%ecx),%esp
 4c:  c3                 ret
 4d:  90                 nop
 4e:  90                 nop
 4f:  90                 nop
 50:  90                 nop
```

```
[...]
```

# Instrumentazione del Codice

- Se è possibile modificare la struttura dei file ELF, è anche possibile alterare il comportamento originale del codice: questa tecnica è chiamata *instrumentazione*.
- Problematiche di questa tecnica:
  - Occorre lavorare al livello di codice macchina: è necessario inserire nel file ELF uno *stream di byte* corrispondenti a particolari istruzioni;
  - Per effettuare instrumentazione trasparente all'utente è necessario preservare la *coerenza dei riferimenti* interni al codice;
  - È altresì necessario poter interpretare il codice originale del programma, per individuare le *giuste posizioni* in cui inserire il codice di instrumentazione.
- Fortemente utilizzata nel *debugging* e nella *vulnerability assessment*.

# Instruction Set i386



Le istruzioni sono quindi di formato variabile  
(con un limite di 16 byte):

```

85 c0          test    %eax,%eax
75 09          jnz     4c
c7 45 ec 00 00 00 00  movl   $0x0,-0x14(%ebp)
eb 59          jmp     a5
8b 45 08       mov     0x8(%ebp),%eax
8d 4c 24 04    lea    0x4(%esp),%ecx
0f b7 40 2e    movzwl 0x2e(%eax),%eax
  
```

Opcode, ModR/M, SIB, Displacement, Immediate

# Instruction Set i386 (2)

$$\left\{ \begin{array}{l} CS: \\ DS: \\ SS: \\ ES: \\ FS: \\ GS: \end{array} \right\} \left[ \begin{array}{l} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{array} \right] + \left[ \begin{array}{l} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{array} \right] * \left\{ \begin{array}{l} 1 \\ 2 \\ 4 \\ 8 \end{array} \right\} + [displacement]$$

- I campi R/M del byte ModR/M e i campi Scale ed Index del byte SIB identificano dei registri;
- I registri sono numerati da 0 a 7 nell'ordine: `eax` (000), `ecx` (001), `edx` (010), `ebx` (011), `esp` (100), `ebp` (101), `esi` (110), `edi` (111).



# Parsing dei file ELF

- Viene scandita la tabella degli header delle sezioni alla ricerca di tutte quelle sezioni contenenti codice (type: PROGBITS, flag: EXECINSTR);
- Ciascuna di queste sezioni viene scandita, byte a byte;
- Tramite una tabella di famiglia di opcode vengono disassemblate le istruzioni, identificando tutte quelle istruzioni di scrittura che abbiano come operando destinazione una locazione di memoria (variabili o memoria allocata dinamicamente);
- L'operando destinazione viene scomposto nelle sue componenti *base*, *indice*, *scala* ed *offset*.

# Generazione della Tabella Istruzioni

- Per aggiungere un overhead minimo al programma originale si attuano due scelte:
  - La routine di monitoring viene scritta direttamente in assembly;
  - Si cerca di evitare di effettuare a run-time l'interpretazione delle istruzioni.
- Durante la fase di parsing, le informazioni di interesse vengono memorizzate all'interno di una tabella:

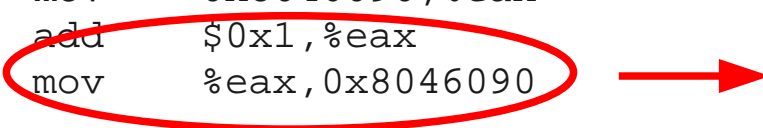
```
struct insn_entry {  
    unsigned long ret_addr;  
    unsigned int size;  
    char flags;  
    char base;  
    char idx;  
    char scala;  
    long offset;  
};
```

- Sulla tabella è possibile effettuare una ricerca binaria in tempo  $O(\log n)$ .

# Aggancio del monitor

- L'aggancio della routine di monitoring avviene preponendo a tutte le istruzioni che effettuano scrittura in memoria una chiamata ad una routine di nome `monitor`;

```
a1 90 60 04 08 mov 0x8046090,%eax  
83 c0 01      add $0x1,%eax  
a3 90 60 04 08 mov %eax,0x8046090
```



```
a1 90 60 04 08 mov 0x8046090,%eax  
83 c0 01      add $0x1,%eax  
e8 fc ff ff ff call monitor  
a3 90 60 04 08 mov %eax,0x8046090
```

- Viene utilizzata una `call` invece di una meno costosa `jump` poiché, tramite il valore di ritorno, è possibile risalire all'istruzione che ha causato la chiamata;

- L'aggiunta di queste chiamate rende necessario il ridimensionamento delle sezioni (secondo le tecniche viste precedentemente) e la correzione delle tabelle di rilocazione.

# Correzione dei riferimenti


- L'inserimento di istruzioni rende inconsistenti i riferimenti tra parti differenti di codice;
- Per questo motivo è necessario:
  - Correggere gli entry point delle funzioni;
  - Correggere tutti i salti
- I salti intra-segmento nell'i386 sono espressi come offset a partire dal valore del registro `eip` al momento dell'esecuzione dell'istruzione;
- Per correggerli, è sufficiente scandire una seconda volta il testo del programma e correggere le destinazioni applicando loro uno shift pari al numero di byte di tutte le istruzioni inserite;

# Correzione dinamica dei salti

- Un particolare tipo di salto (*indirect branch*, salto a registro) permette di specificare la destinazione del salto tramite un valore memorizzato in un registro o in un'area di memoria;
- La semantica di questa istruzione *dipende dal flusso di esecuzione*: non è possibile correggerla tramite un'istrumentazione statica;
- Queste istruzioni vengono trattate come le scritture in memoria: le istruzioni vengono sostituite con una chiamata ad una routine di correzione dinamica (`correct_branch`) che, tramite le informazioni in due tabelle fa effettuare un salto corretto.

```
8b 04 95 2c 00 00 00 mov 0x2c(,%edx,4),%eax
ff e0 jmp *%eax
```

```
8b 04 95 2c 00 00 00 mov 0x2c(,%edx,4),%eax
e8 fc ff ff ff call correct_branch
e9 00 00 00 00 jmp ?? ?? ?? ??
```



# Esecuzione del tracciamento

```
...  
call monitor  
mov %eax, i  
...
```

applicazione

*CPU*

EAX: ?????????????? ESI: ??????????????

EBX: ?????????????? EDI: ??????????????

ECX: ?????????????? EBP: ??????????????

EDX: ?????????????? ESP: ??????????????

# Esecuzione del tracciamento

```
...  
call monitor  
mov %eax, i  
...
```

applicazione



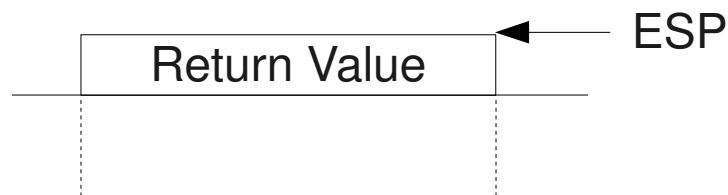
monitor

CPU

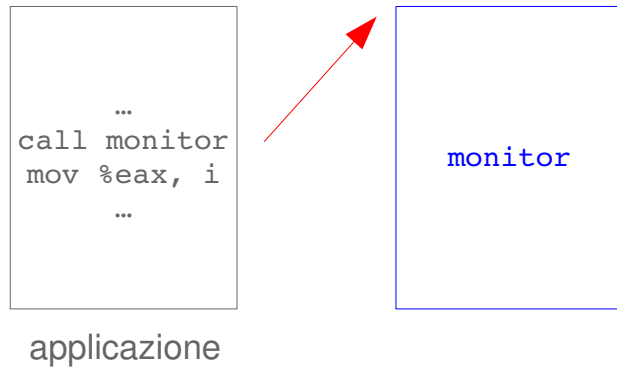
```
EAX: ?????????????? ESI: ??????????????  
EBX: ?????????????? EDI: ??????????????  
ECX: ?????????????? EBP: ??????????????  
EDX: ?????????????? ESP: ??????????????
```

monitor:

```
push    %eax  
push    %ecx  
push    %edx  
push    %ebx  
mov     %esp, %eax  
sub     $4, %esp  
add     $16, %eax  
mov     %eax, (%esp)  
push    %ebp  
push    %esi  
push    %edi  
pushfw  
mov     14(%esp), %ebp  
sub     $4, %ebp  
mov     4(%ebp), %esi
```



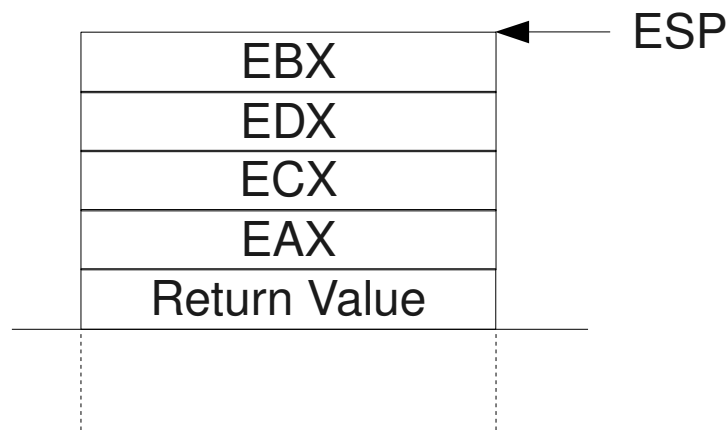
# Esecuzione del tracciamento



*CPU*

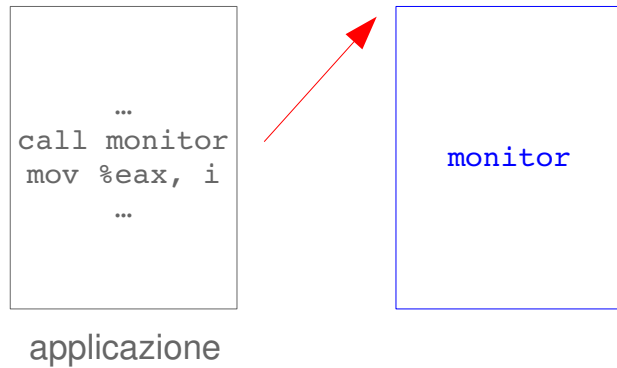
```
EAX: ??????????????  ESI: ??????????????  
EBX: ??????????????  EDI: ??????????????  
ECX: ??????????????  EBP: ??????????????  
EDX: ??????????????  ESP: ??????????????
```

```
monitor:  
    push    %eax  
    push    %ecx  
    push    %edx  
    push    %ebx  
    mov     %esp, %eax  
    sub     $4, %esp  
    add     $16, %eax  
    mov     %eax, (%esp)  
    push    %ebp  
    push    %esi  
    push    %edi  
    pushfw  
    mov     14(%esp), %ebp  
    sub     $4, %ebp  
    mov     4(%ebp), %esi
```





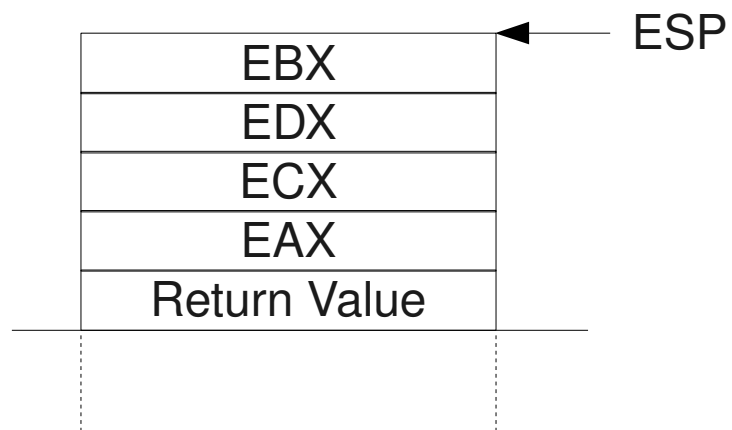
# Esecuzione del tracciamento



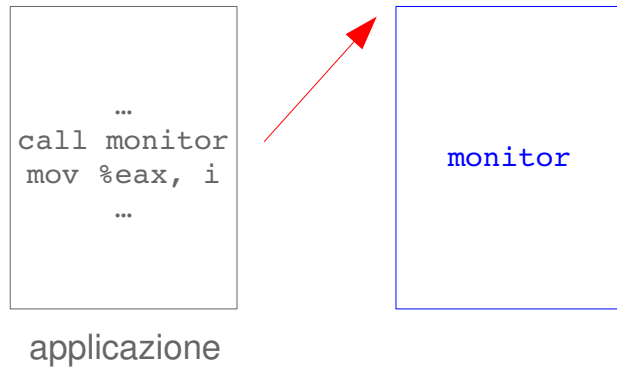
*CPU*

```
EAX: esp corrente   ESI: ??????????????  
EBX: ?????????????? EDI: ??????????????  
ECX: ?????????????? EBP: ??????????????  
EDX: ?????????????? ESP: ??????????????
```

```
monitor:  
    push    %eax  
    push    %ecx  
    push    %edx  
    push    %ebx  
    mov     %esp, %eax  
    sub     $4, %esp  
    add     $16, %eax  
    mov     %eax, (%esp)  
    push    %ebp  
    push    %esi  
    push    %edi  
    pushfw  
    mov     14(%esp), %ebp  
    sub     $4, %ebp  
    mov     4(%ebp), %esi
```



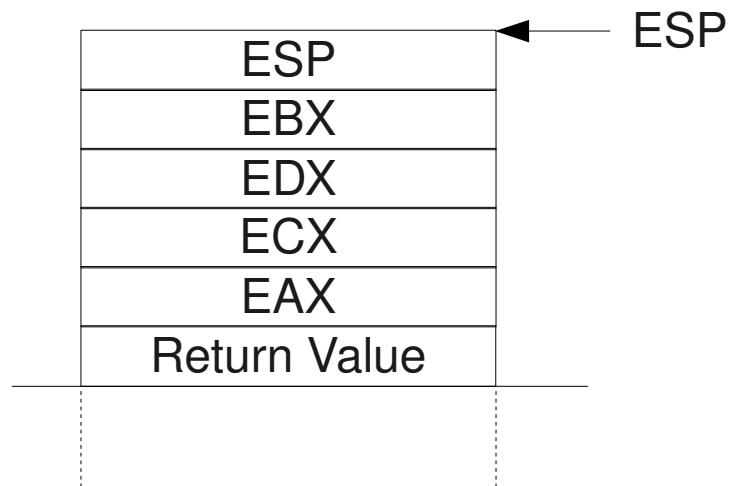
# Esecuzione del tracciamento



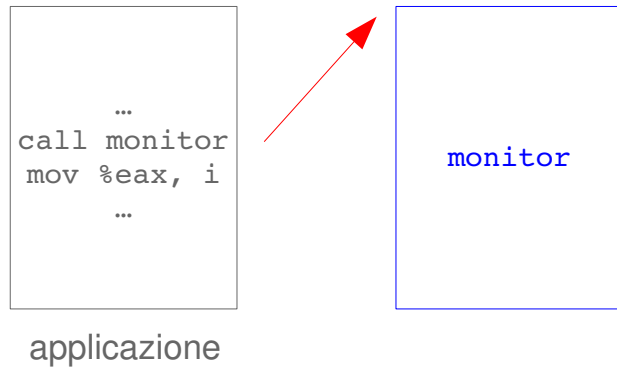
CPU

```
EAX: esp originale   ESI: ??????????????  
EBX: ?????????????? EDI: ??????????????  
ECX: ?????????????? EBP: ??????????????  
EDX: ?????????????? ESP: ??????????????
```

```
monitor:  
    push    %eax  
    push    %ecx  
    push    %edx  
    push    %ebx  
    mov     %esp, %eax  
    sub     $4, %esp  
    add     $16, %eax  
    mov     %eax, (%esp)  
    push    %ebp  
    push    %esi  
    push    %edi  
    pushfw  
    mov     14(%esp), %ebp  
    sub     $4, %ebp  
    mov     4(%ebp), %esi
```

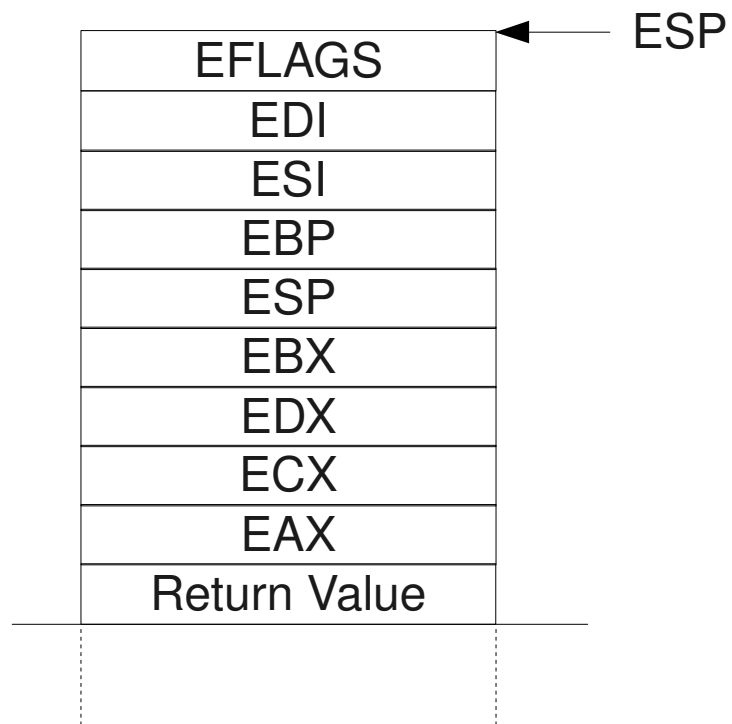


# Esecuzione del tracciamento



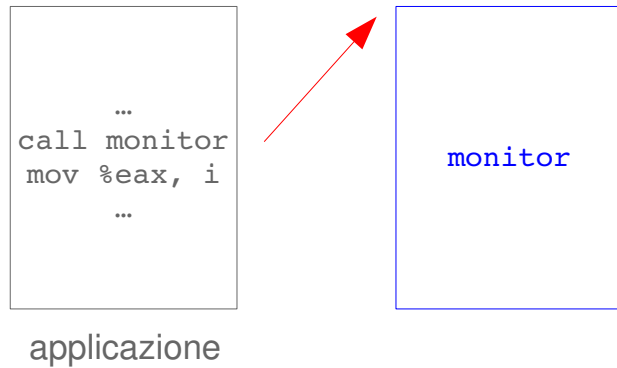
*CPU*

```
EAX: esp originale   ESI: ??????????????  
EBX: ?????????????? EDI: ??????????????  
ECX: ?????????????? EBP: ??????????????  
EDX: ?????????????? ESP: ??????????????
```



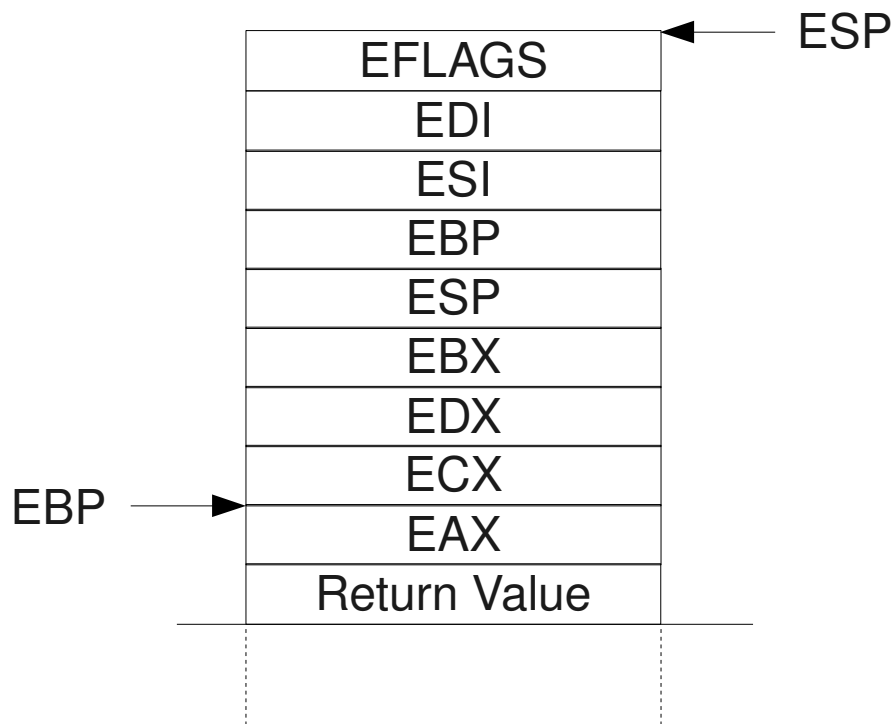
```
monitor:  
    push    %eax  
    push    %ecx  
    push    %edx  
    push    %ebx  
    mov     %esp, %eax  
    sub     $4, %esp  
    add     $16, %eax  
    mov     %eax, (%esp)  
    push   %ebp  
    push   %esi  
    push   %edi  
    pushfw  
    mov     14(%esp), %ebp  
    sub     $4, %ebp  
    mov     4(%ebp), %esi
```

# Esecuzione del tracciamento



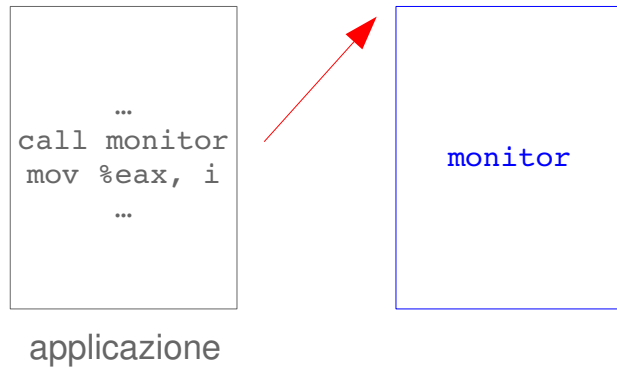
*CPU*

```
EAX: esp originale    ESI: ??????????????  
EBX: ?????????????? EDI: ??????????????  
ECX: ?????????????? EBP: indirizzo eax orig.  
EDX: ?????????????? ESP: ??????????????
```



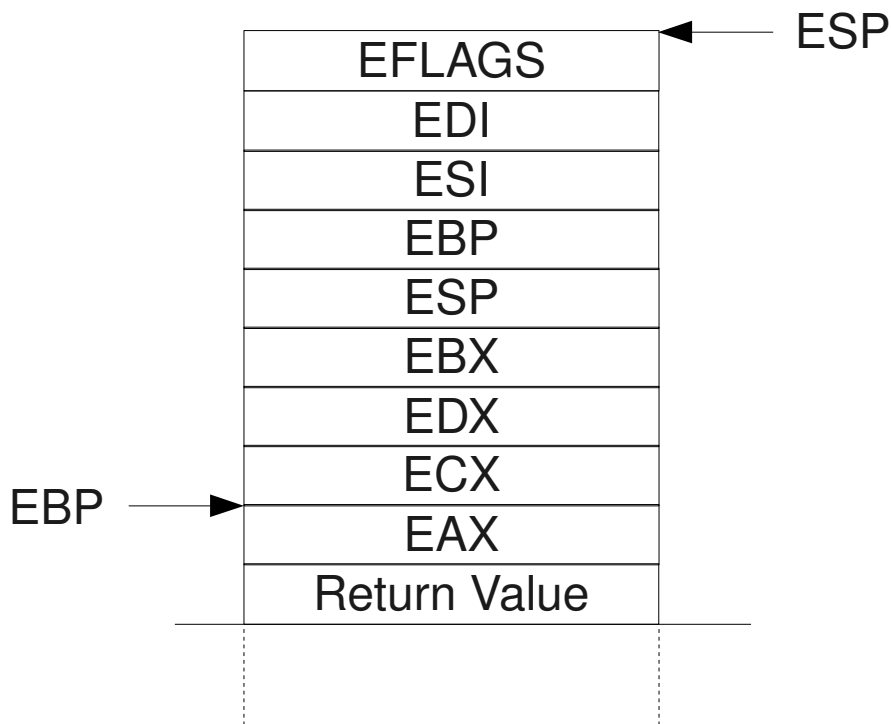
```
monitor:  
    push    %eax  
    push    %ecx  
    push    %edx  
    push    %ebx  
    mov     %esp, %eax  
    sub     $4, %esp  
    add     $16, %eax  
    mov     %eax, (%esp)  
    push    %ebp  
    push    %esi  
    push    %edi  
    pushfw  
    mov     14(%esp), %ebp  
    sub     $4, %ebp  
    mov     4(%ebp), %esi
```

# Esecuzione del tracciamento



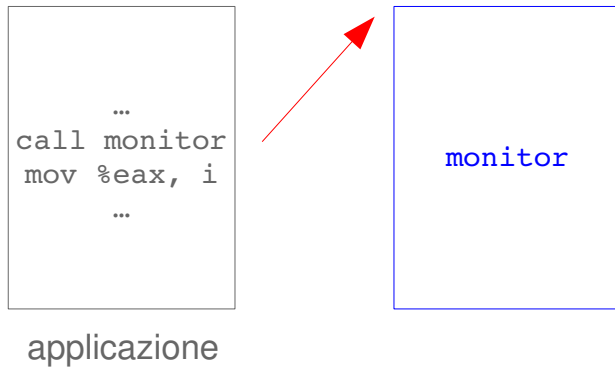
*CPU*

EAX: <i>esp originale</i>	ESI: <i>chiave di ricerca</i>
EBX: ????????????????	EDI: ????????????????
ECX: ????????????????	EBP: <i>indirizzo eax orig.</i>
EDX: ????????????????	ESP: ????????????????



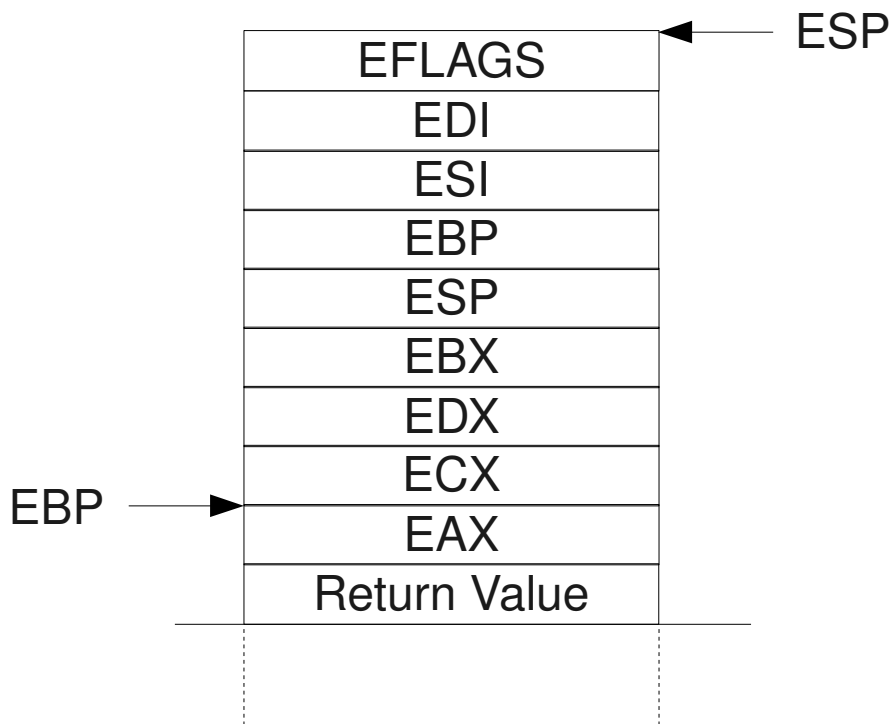
```
monitor:  
    push    %eax  
    push    %ecx  
    push    %edx  
    push    %ebx  
    mov     %esp, %eax  
    sub     $4, %esp  
    add     $16, %eax  
    mov     %eax, (%esp)  
    push    %ebp  
    push    %esi  
    push    %edi  
    pushfw  
    mov     14(%esp), %ebp  
    sub     $4, %ebp  
    mov     4(%ebp), %esi
```

# Esecuzione del tracciamento



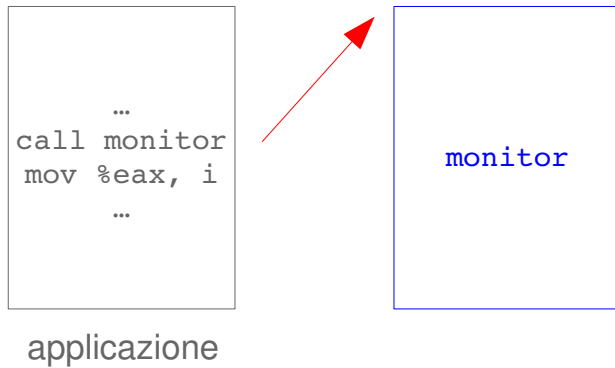
*CPU*

EAX: *esp originale*      ESI: *chiave di ricerca*  
 EBX: *low*                      EDI: *????????????????*  
 ECX: *high*                     EBP: *indirizzo eax orig.*  
 EDX: *????????????????*      ESP: *????????????????*



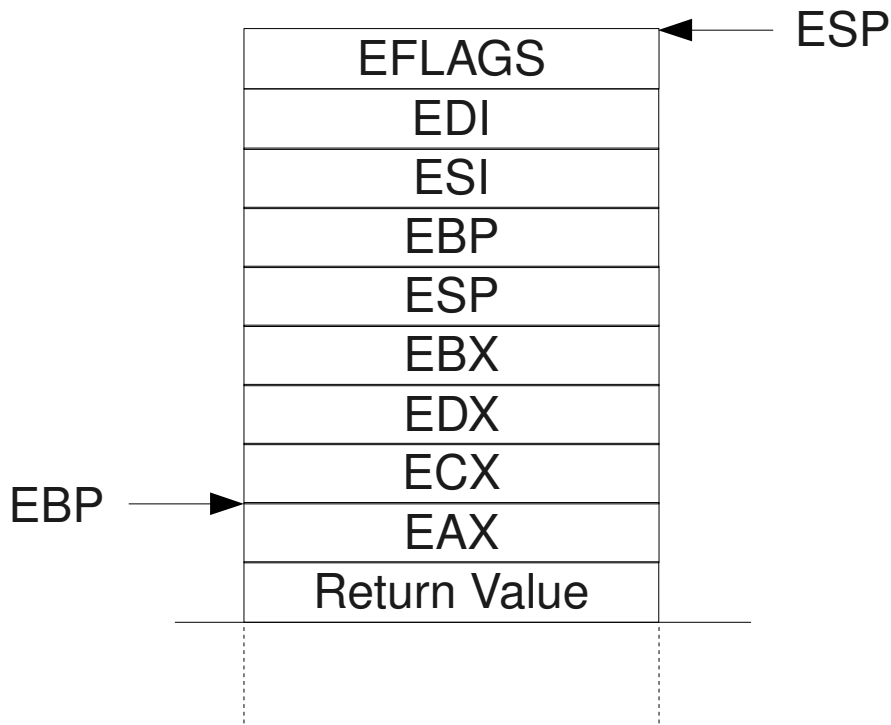
```
monitor:
    xor    %ebx, %ebx
    mov    $DIM, %ecx
    jmp    .Cerca
.HighHalf: lea    0x1(%edx), %ebx
           cmp    %ecx, %ebx
           jae   .Trovato
.Cerca:  lea    (%ecx,%ebx,1), %edx
           shr    %edx
           mov    %edx, %eax
           shl    $0x4,%eax
           cmp    %esi, insn_table(%eax)
           jb    .HighHalf
.LowHalf: mov    %edx, %ecx
           cmp    %ecx, %ebx
           jb    .Cerca
.Trovato:
```

# Esecuzione del tracciamento



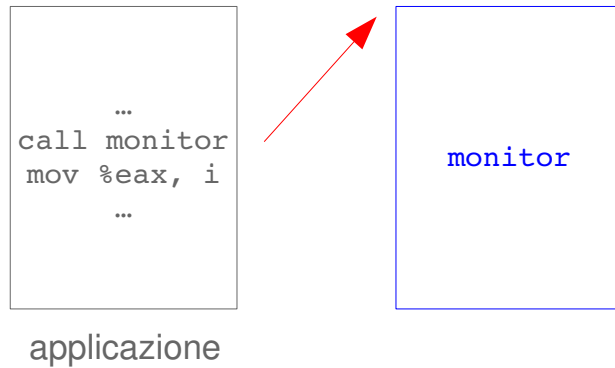
**CPU**

EAX: <i>offset nella tab.</i>	ESI: <i>chiave di ricerca</i>
EBX: <i>low</i>	EDI: <i>????????????????</i>
ECX: <i>high</i>	EBP: <i>indirizzo eax orig.</i>
EDX: <i>mediano</i>	ESP: <i>????????????????</i>



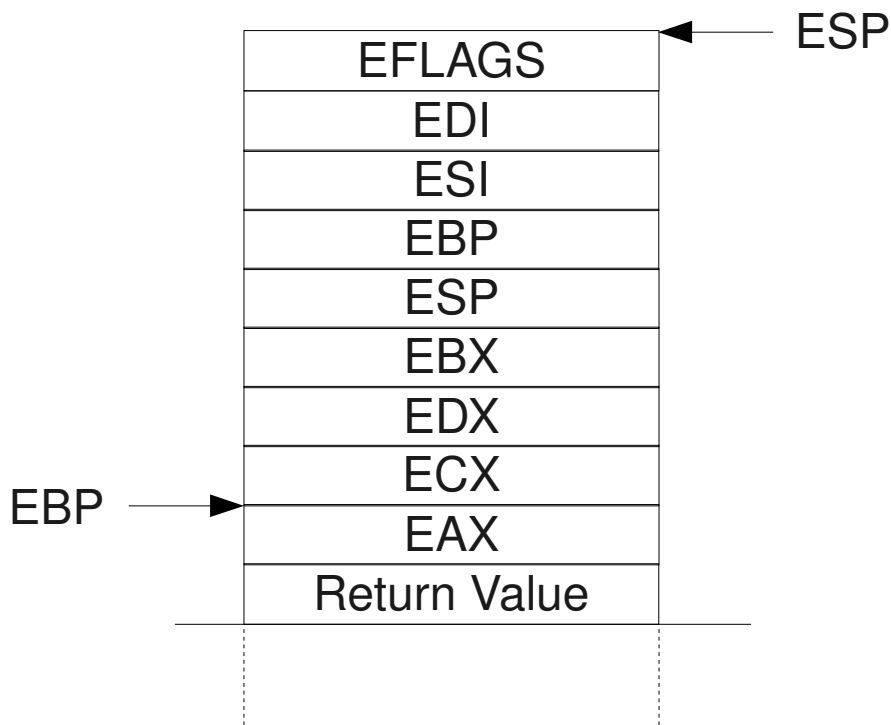
```
monitor:
    xor    %ebx, %ebx
    mov    $DIM, %ecx
    jmp    .Cerca
.HighHalf: lea    0x1(%edx), %ebx
    cmp    %ecx, %ebx
    jae    .Trovato
.Cerca:  lea    (%ecx,%ebx,1), %edx
    shr    %edx
    mov    %edx, %eax
    shl    $0x4,%eax
    cmp    %esi, insn_table(%eax)
    jb    .HighHalf
.LowHalf: mov    %edx, %ecx
    cmp    %ecx, %ebx
    jb    .Cerca
.Trovato:
```

# Esecuzione del tracciamento



**CPU**

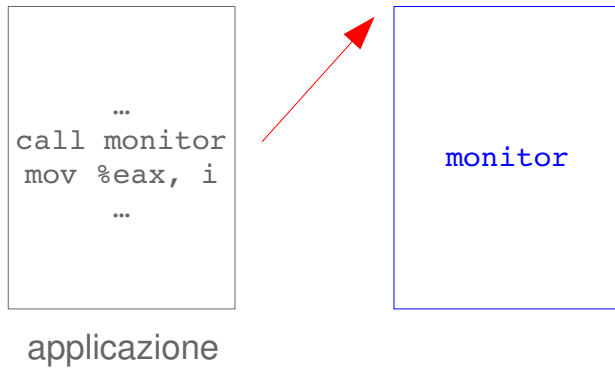
EAX: <i>offset nella tab.</i>	ESI: <i>chiave di ricerca</i>
EBX: <i>low</i>	EDI: <i>????????????????</i>
ECX: <i>high</i>	EBP: <i>indirizzo eax orig.</i>
EDX: <i>nuovo low</i>	ESP: <i>????????????????</i>



```
monitor:
    xor    %ebx, %ebx
    mov    $DIM, %ecx
    jmp    .Cerca
.HighHalf: lea    0x1(%edx), %ebx
           cmp    %ecx, %ebx
           jae   .Trovato
.Cerca:  lea    (%ecx,%ebx,1), %edx
           shr   %edx
           mov   %edx, %eax
           shl   $0x4,%eax
           cmp   %esi, insn_table(%eax)
           jb   .HighHalf
.LowHalf: mov   %edx, %ecx
           cmp   %ecx, %ebx
           jb   .Cerca
.Trovato:
```

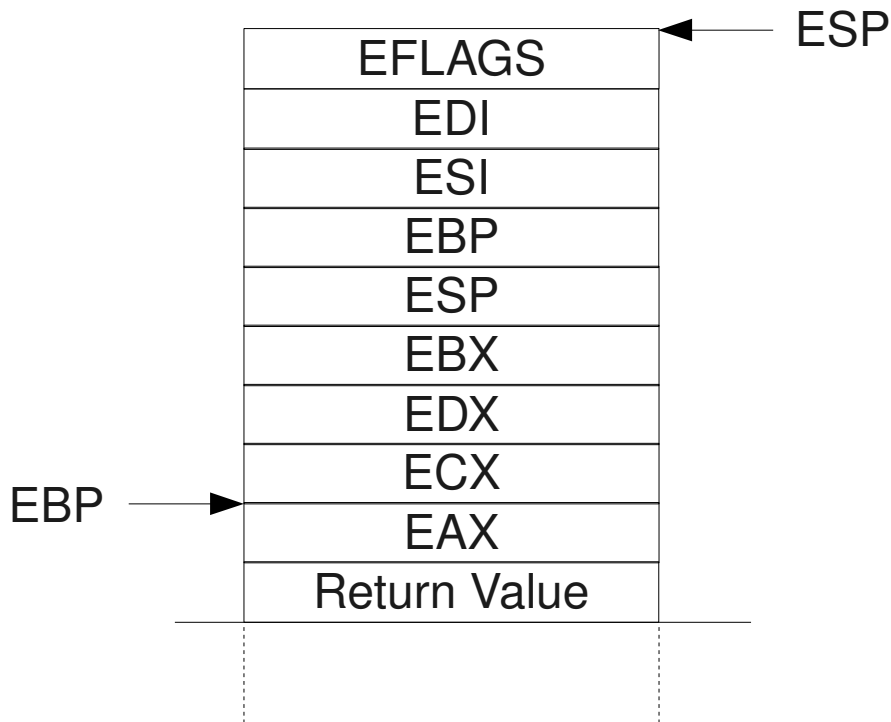


# Esecuzione del tracciamento



**CPU**

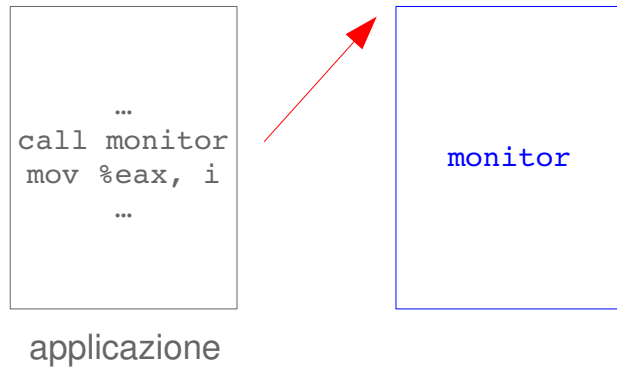
EAX: <i>offset nella tab.</i>	ESI: <i>chiave di ricerca</i>
EBX: <i>low</i>	EDI: <i>????????????????</i>
ECX: <i>nuovo high</i>	EBP: <i>indirizzo eax orig.</i>
EDX: <i>mediano</i>	ESP: <i>????????????????</i>



```

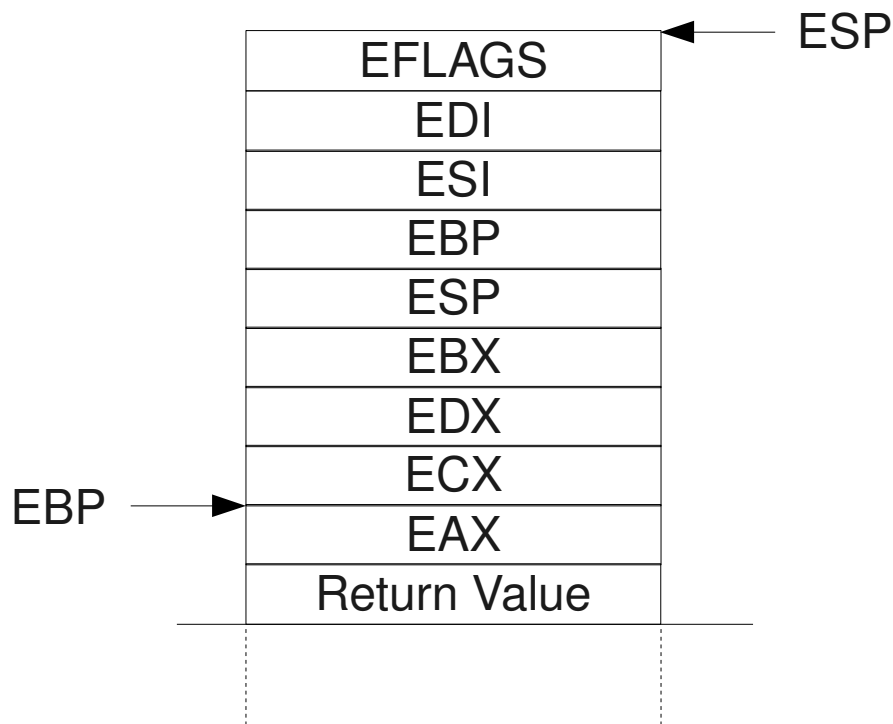
monitor:
    xor    %ebx, %ebx
    mov    $DIM, %ecx
    jmp    .Cerca
.HighHalf: lea    0x1(%edx), %ebx
            cmp    %ecx, %ebx
            jae   .Trovato
.Cerca:  lea    (%ecx,%ebx,1), %edx
            shr    %edx
            mov    %edx, %eax
            shl    $0x4,%eax
            cmp    %esi, insn_table(%eax)
            jb    .HighHalf
.LowHalf: mov    %edx, %ecx
            cmp    %ecx, %ebx
            jb    .Cerca
.Trovato:
    
```

# Esecuzione del tracciamento



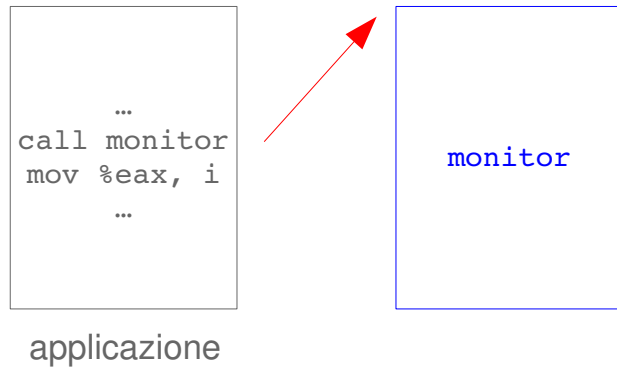
*CPU*

EAX: <i>campo flags</i>	ESI: <i>chiave di ricerca</i>
EBX: <i>low</i>	EDI: <i>???????????????</i>
ECX: <i>nuovo high</i>	EBP: <i>indirizzo eax orig.</i>
EDX: <i>offset tabella</i>	ESP: <i>???????????????</i>



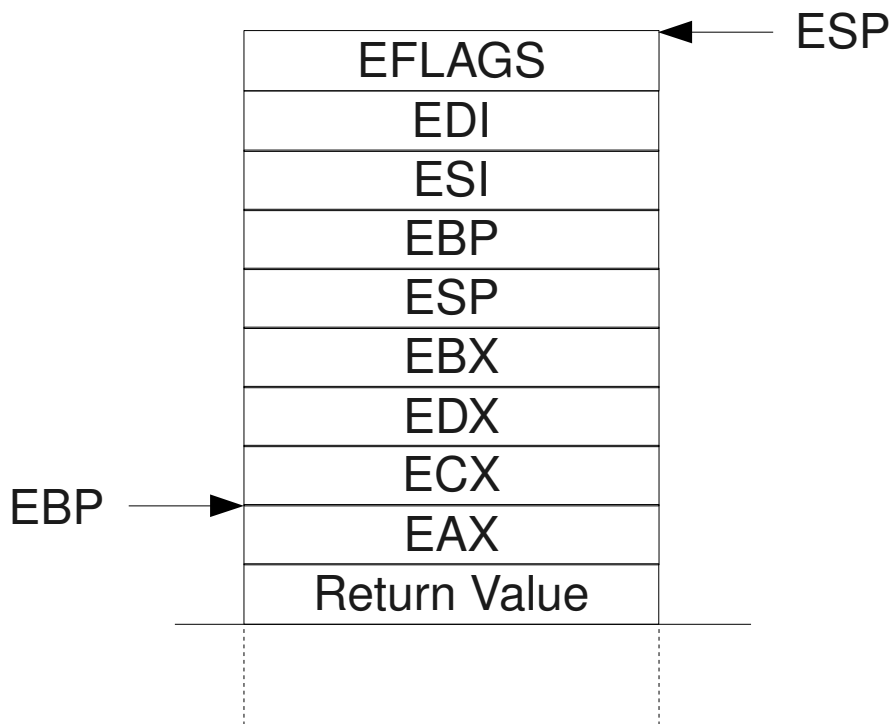
```
monitor:  
    lea    (,%ecx,4), %edx  
    shl   $0x2, %edx  
    movsbl insn_table+8(%edx),%eax  
    xor   %edi, %edi  
    testb $4, %al  
    jz    .NoIndex  
    movsbl insn_table+10(%edx),%ecx  
    negl  %ecx  
    movl  (%ebp, %ecx, 4), %edi  
    movsbl insn_table+11(%edx),%ecx  
    imul  %ecx, %edi
```

# Esecuzione del tracciamento



CPU

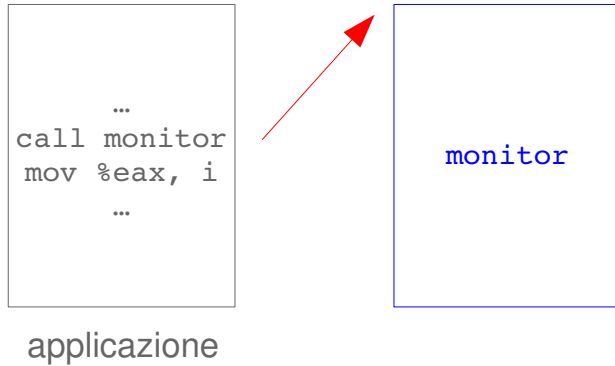
EAX: *campo flags*      ESI: *chiave di ricerca*  
EBX: *low*                EDI: *idx*  
ECX: *- reg. indice*      EBP: *indirizzo eax orig.*  
EDX: *offset tabella*    ESP: *???????????????*



monitor:

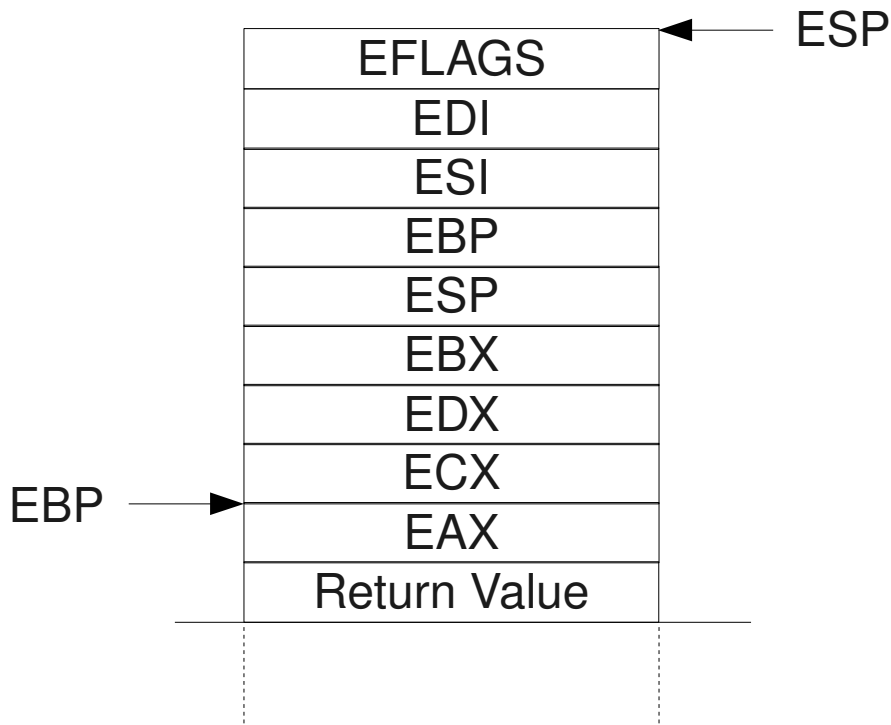
```
lea    (,%ecx,4), %edx
shl    $0x2, %edx
movsbl insn_table+8(%edx),%eax
xor    %edi, %edi
testb  $4, %al
jz     .NoIndex
movsbl insn_table+10(%edx),%ecx
negl   %ecx
movl   (%ebp, %ecx, 4), %edi
movsbl insn_table+11(%edx),%ecx
imul  %ecx, %edi
```

# Esecuzione del tracciamento



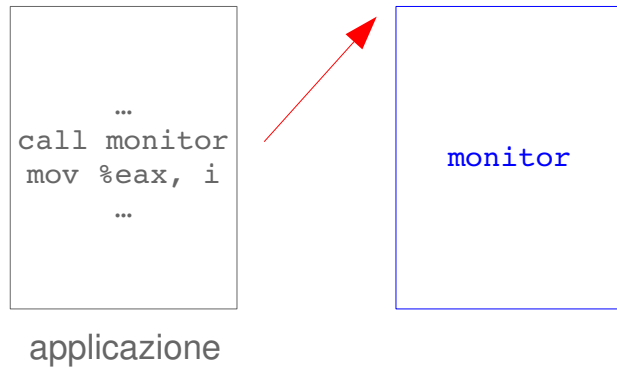
**CPU**

EAX: <i>campo flags</i>	ESI: <i>chiave di ricerca</i>
EBX: <i>low</i>	EDI: <i>idx * scala</i>
ECX: <i>scala</i>	EBP: <i>indirizzo eax orig.</i>
EDX: <i>offset tabella</i>	ESP: <i>???????????????</i>



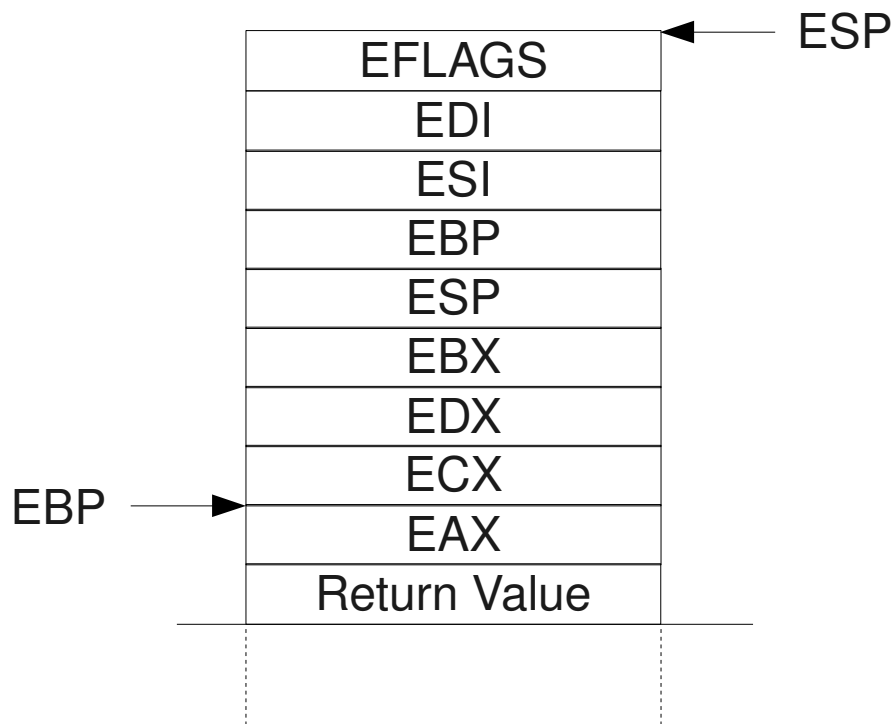
```
monitor:  
    lea    (,%ecx,4), %edx  
    shl   $0x2, %edx  
    movsbl insn_table+8(%edx),%eax  
    xor   %edi, %edi  
    testb $4, %al  
    jz    .NoIndex  
    movsbl insn_table+10(%edx),%ecx  
    negl  %ecx  
    movl  (%ebp, %ecx, 4), %edi  
    movsbl insn_table+11(%edx),%ecx  
    imul  %ecx, %edi
```

# Esecuzione del tracciamento



**CPU**

EAX: <i>campo flags</i>	ESI: <i>chiave di ricerca</i>
EBX: <i>low</i>	EDI: <i>base + idx * scala</i>
ECX: <i>- reg. base</i>	EBP: <i>indirizzo eax orig.</i>
EDX: <i>offset tabella</i>	ESP: <i>???????????????</i>

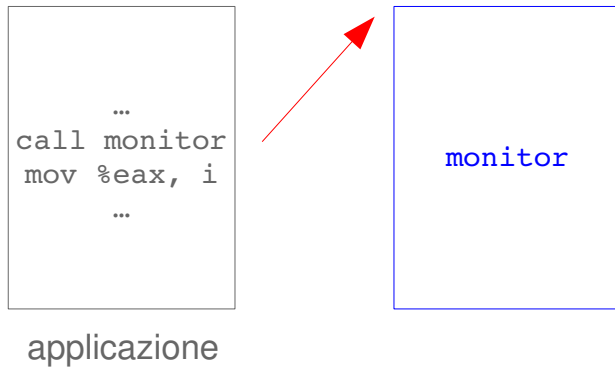


```
monitor:
    .NoIndex:
        testb    $2, %al
        jz      .NoBase
        movsbl  insn_table+9(%edx), %ecx
        negl   %ecx
        addl   (%ebp, %ecx, 4), %edi

    .NoBase:
        add    insn_table+12(%edx), %edi
        movsbl insn_table+4(%edx), %esi

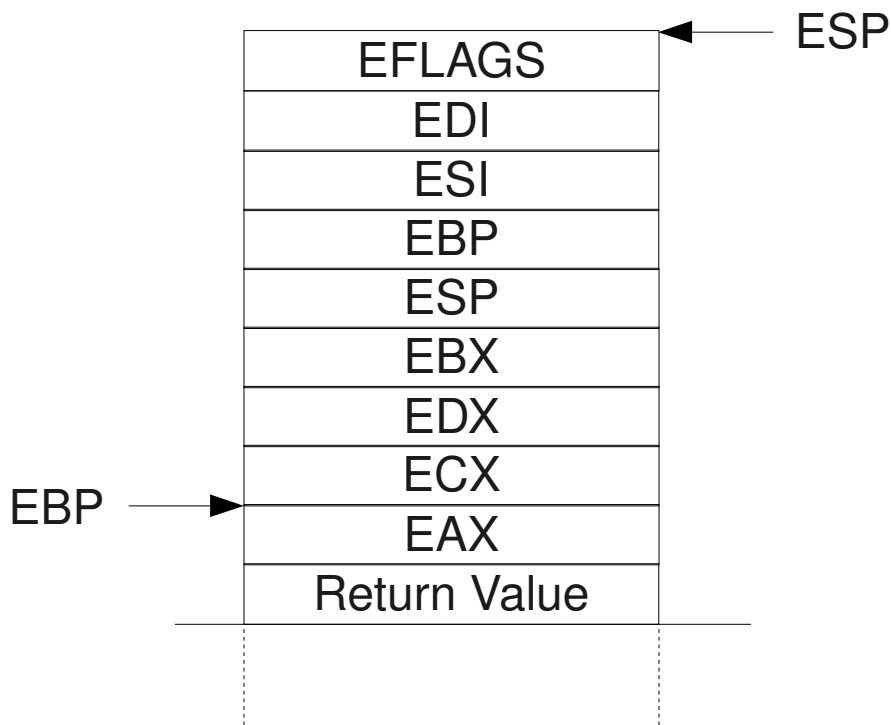
        push  %esi
        push  %edi
        call  dirty_mem
        addl  $8, %esp
```

# Esecuzione del tracciamento



**CPU**

EAX: campo flags	ESI: <i>taglia</i>
EBX: low	EDI: <i>bs + idx * scl + off</i>
ECX: - reg. base	EBP: indirizzo eax orig.
EDX: mediano	ESP: ????????????????



```
monitor:  
    .NoIndex:  
        testb    $2, %al  
        jz      .NoBase  
        movsbl  insn_table+9(%edx), %ecx  
        negl   %ecx  
        addl   (%ebp, %ecx, 4), %edi  
  
    .NoBase:  
        add    insn_table+12(%edx), %edi  
        movsbl insn_table+4(%edx), %esi  
  
        push  %esi  
        push  %edi  
        call  dirty_mem  
        addl  $8, %esp
```

# Esecuzione del tracciamento

```
...
call monitor
mov %eax, i
...
```

applicazione

```
monitor
```

```
dirty_mem
```

**CPU**

EAX: <i>campo flags</i>	ESI: <i>taglia</i>
EBX: <i>low</i>	EDI: <i>bs + idx * scl + off</i>
ECX: <i>- reg. base</i>	EBP: <i>indirizzo eax orig.</i>
EDX: <i>mediano</i>	ESP: <i>???????????????</i>

Destinazione
Taglia
EFLAGS
EDI
ESI
EBP
ESP
EBX
EDX
ECX
EAX
Return Value

EBP

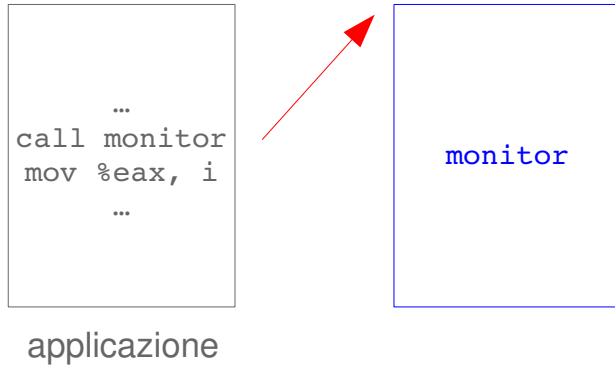
ESP

```
monitor:
.NoIndex:
    testb    $2, %al
    jz      .NoBase
    movsbl  insn_table+9(%edx), %ecx
    negl    %ecx
    addl    (%ebp, %ecx, 4), %edi

.NoBase:
    add     insn_table+12(%edx), %edi
    movsbl  insn_table+4(%edx), %esi

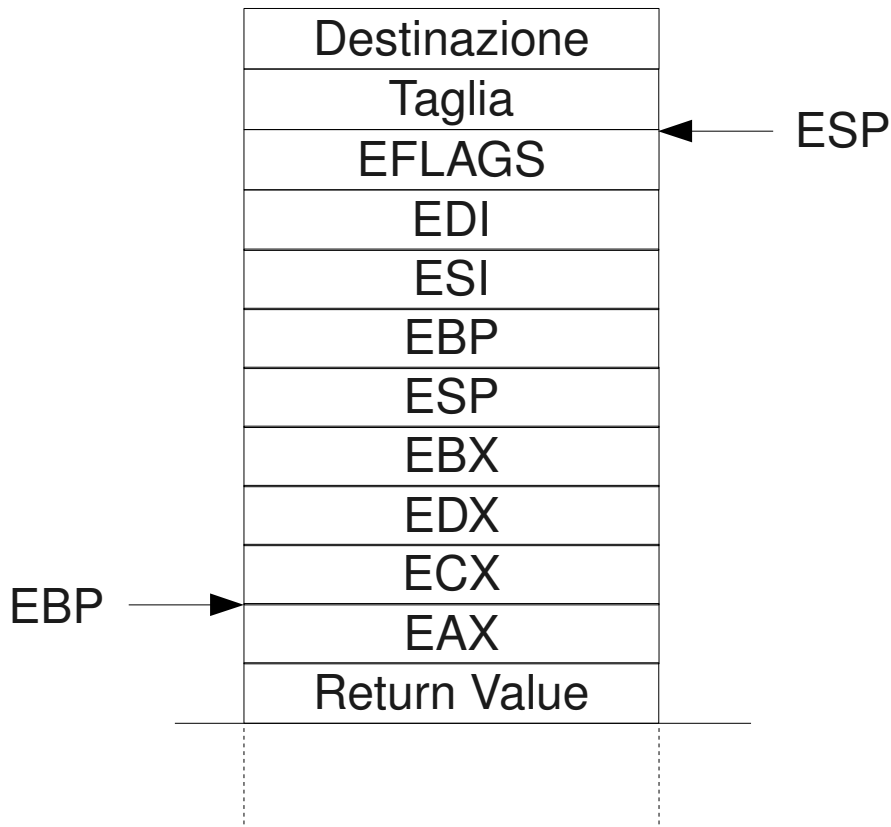
    push   %esi
    push   %edi
    call   dirty_mem
    addl   $8, %esp
```

# Esecuzione del tracciamento



```
CPU
```

EAX: ?????????????? ESI: ??????????????  
 EBX: ?????????????? EDI: ??????????????  
 ECX: ?????????????? EBP: ??????????????  
 EDX: ?????????????? ESP: ??????????????



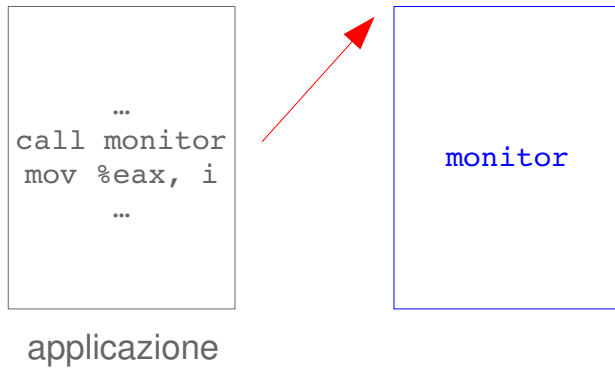
```
monitor:
    .NoIndex:
        testb    $2, %al
        jz      .NoBase
        movsbl  insn_table+9(%edx), %ecx
        negl   %ecx
        addl   (%ebp, %ecx, 4), %edi

    .NoBase:
        add     insn_table+12(%edx), %edi
        movsbl  insn_table+4(%edx), %esi

        push   %esi
        push   %edi
        call   dirty_mem
        addl   $8, %esp
```

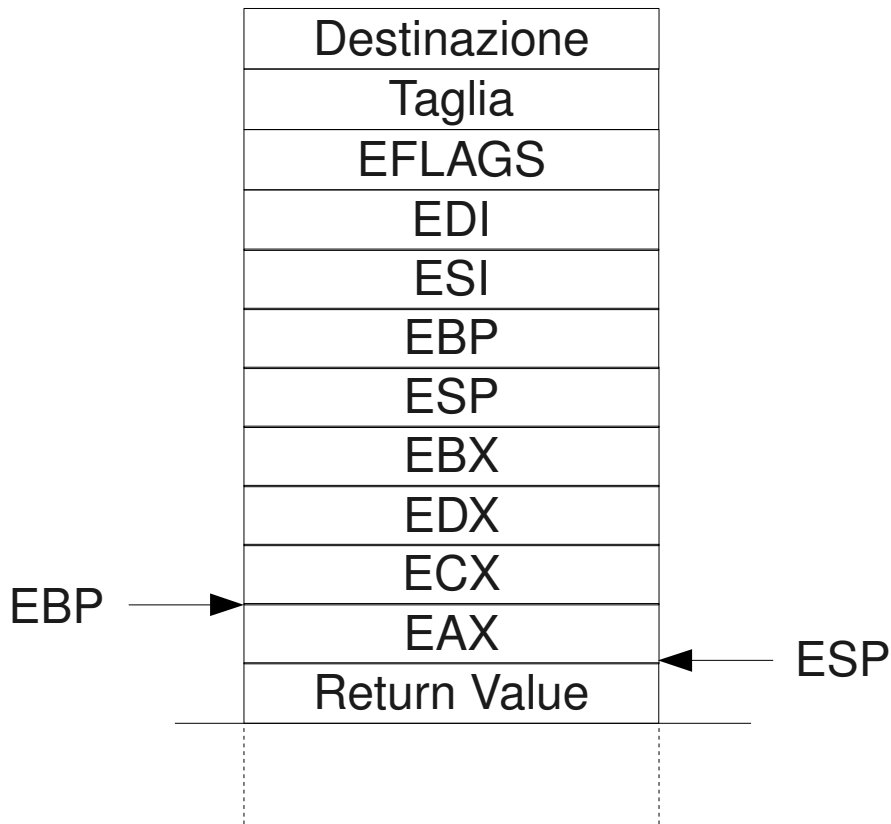


# Esecuzione del tracciamento



CPU

EAX: <i>eax originale</i>	ESI: <i>esi originale</i>
EBX: <i>ebx originale</i>	EDI: <i>edi originale</i>
ECX: <i>ecx originale</i>	EBP: <i>ebp originale</i>
EDX: <i>edx originale</i>	ESP: <i>???????????????</i>



```
monitor:  
    popfw  
    pop    %edi  
    pop    %esi  
    pop    %ebp  
    add    $4, %esp  
    pop    %ebx  
    pop    %edx  
    pop    %ecx  
    pop    %eax  
    ret
```

# Esecuzione del tracciamento

```
...  
call monitor  
mov %eax, i  
...
```

applicazione

← controllo

*CPU*

EAX: <i>eax originale</i>	ESI: <i>esi originale</i>
EBX: <i>ebx originale</i>	EDI: <i>edi originale</i>
ECX: <i>ecx originale</i>	EBP: <i>ebp originale</i>
EDX: <i>edx originale</i>	ESP: <i>esp originale</i>

monitor:

```
popfw  
pop    %edi  
pop    %esi  
pop    %ebp  
add    $4, %esp  
pop    %ebx  
pop    %edx  
pop    %ecx  
pop    %eax  
ret
```

# Riepilogo

