x86

Gabriel Laskar <gabriel@lse.epita.fr>

LSE Security System

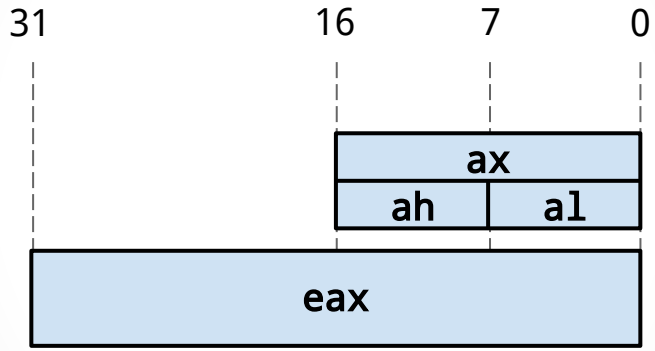
Multiple kind of x86 registers

- General purpose registers
- Segment registers
- FLAGS
- Control & Memory registers

General purpose registers

- `%eax`, `%ebx`, `%ecx`, `%edx`
- `%esi`, `%edi`
- `%esp`, `%ebp`
- `%eip`

Register Aliases



Flags register

- flags, eflags
- pushf, popf
- contains information about execution of the last instruction

LSE Security System

Flags

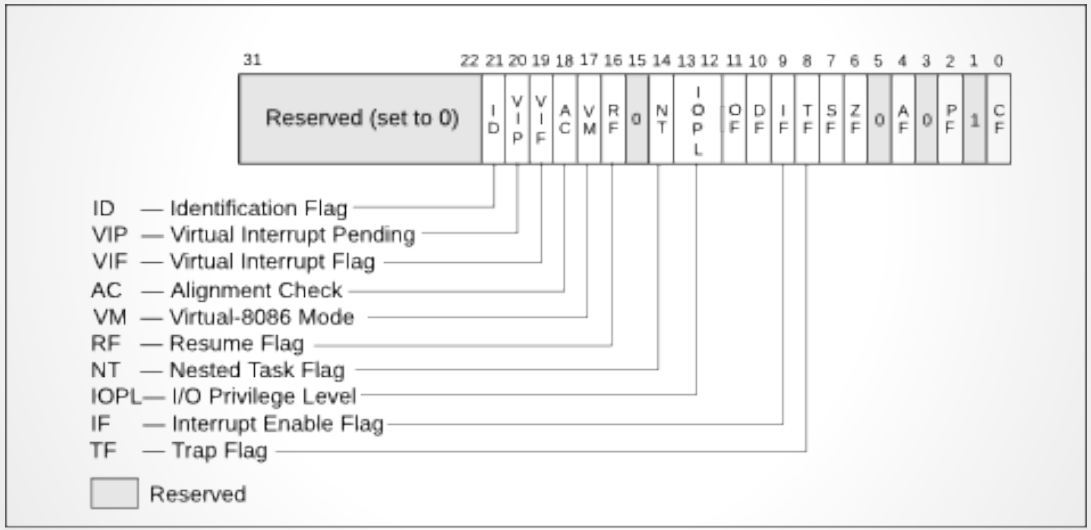
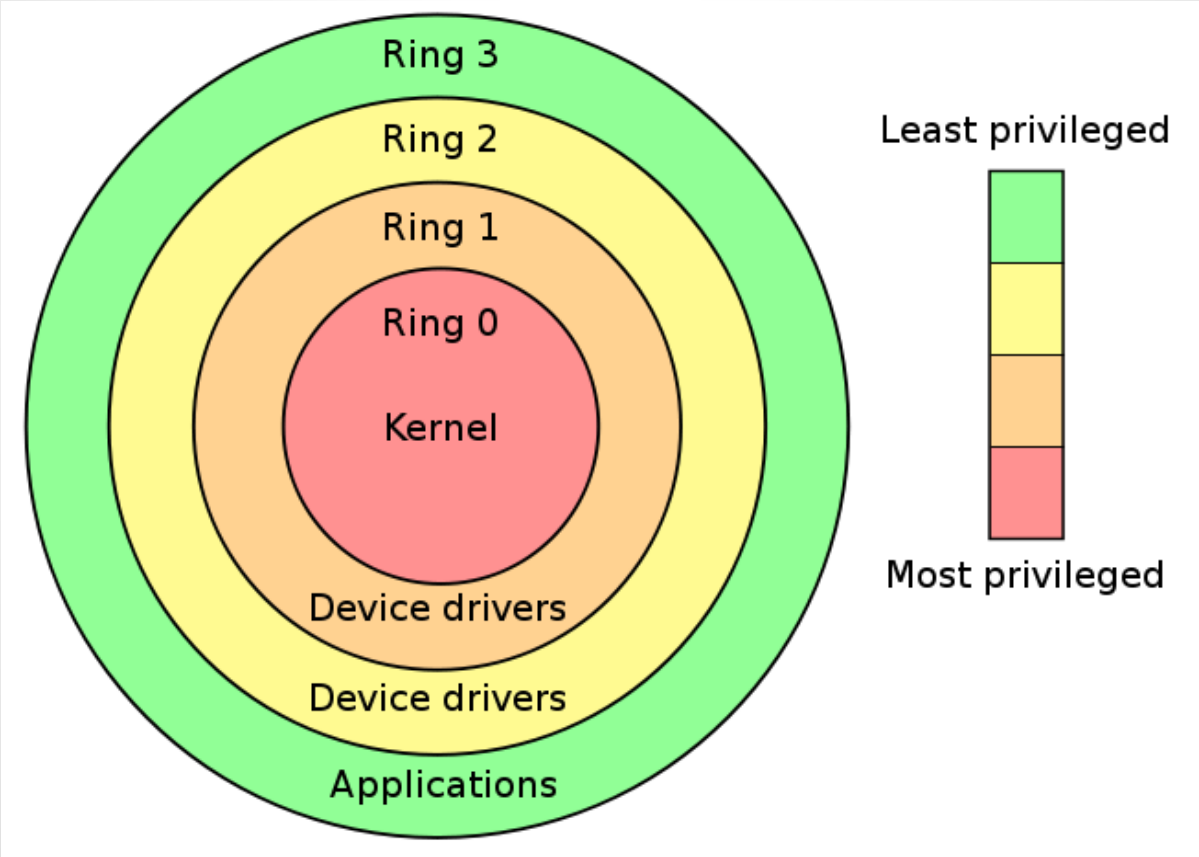


Figure 2-5. System Flags in the EFLAGS Register

Rings


Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged

Control registers

- cr0 : system control flags
- cr2 : page fault linear address
- cr3 : address space address
- cr4 : architecture extensions
- cr8 : Task Priority Register

Control Registers

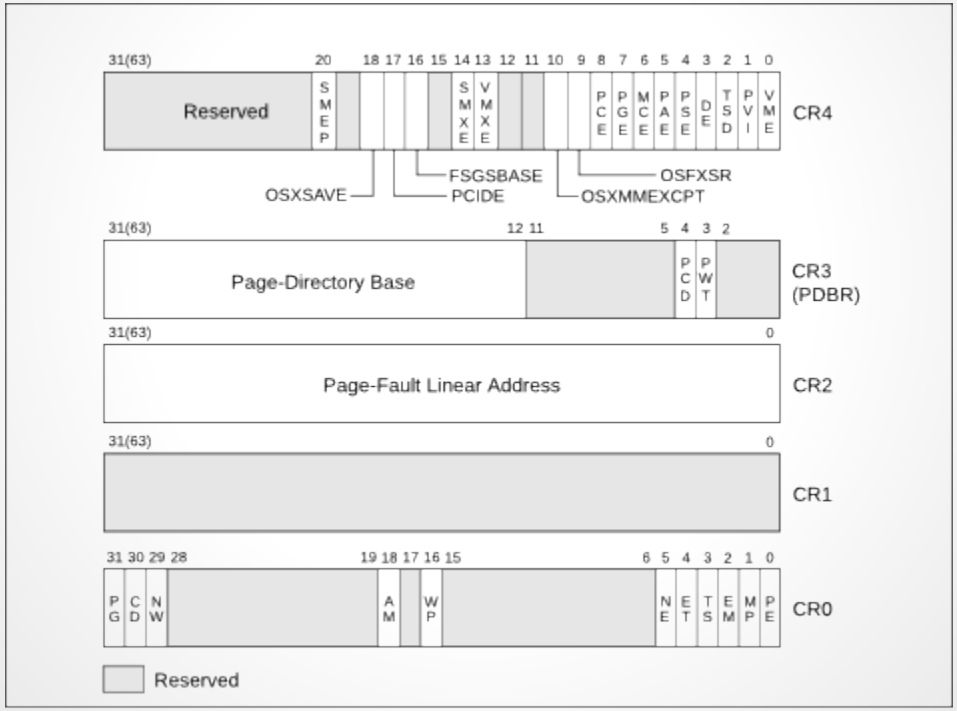


Figure 2-7. Control Registers

LSE Security System

%cr0

- Paging (bit 31)
- Cache Disable (bit 30)
- Not Write-through (bit)
- Alignment Mask (bit 18)
- Write Protect (bit 16)
- Numeric Error (bit 5)
- Extension Type (bit 4)
- Task Switched (bit 3)
- Emulation (bit 2)
- Monitor Coprocessor (bit 1)
- Protection Enable (bit 0)

LSE Security System

Calling Conventions

- Lots of different ways to call a function
- here we focus on linux

http://stackoverflow.com/questions/2535989/what-are-the-calling-conventions-for-unix-linux-system-calls-on-x86-64

LSE Security System

x86_32 : calling functions

- on x86_32 :
  - arguments on the stack, in reverse order
  - return value in %eax
  - %eax, %ecx, %edx saved by caller
  - stack must be 16-byte aligned

x86_32 : syscalls

- %ecx, %edx, %edi and %ebp
- instruction `int $0x80`
- The number of the syscall has to be passed in register %eax
- %eax contains the result of the system-call

ASM Inline

```
__asm ("mov $0, %eax\n");

__asm ("[your assembly code]"
    : output operands /* optional */
    : input operands /* optional */
    : clobber list /*optional*/);
```

LSE Security System

ASM Inline - next level

```
__asm volatile ("mov $0, %eax\n");
```

LSE Security System

ASM Inline - clobber list

- Different output/input constraints
  - `m` : memory operand
  - `r` : register operand
- Constraint modifiers
  - `=` : Write Only
  - `+` : Read/Write
- Different clobbers
  - `memory`
  - register names

LSE Security System

ASM Inline - example

```
__asm volatile ("outb %0, %1\n\t"
        : /* no output */
        : "a" (val), "d" (port));

__asm volatile ("inb %1, %0\n\t"
        : "=&a" (res)
        : "d" (port));
```

Bitfields

```
struct bitfields {
    unsigned int field_a : 1; /* max value is 0b1 */
    unsigned int field_b : 2; /* max value is 0b11 */
    unsigned int field_c : 5; /* max value is 0x1f */
};

sizeof(struct bitfields) == sizeof(char)
```

LSE Security System

Packed structs

```
struct foo {
        unsigned char           a;              /*     0     1 */

        /* XXX 3 bytes hole, try to pack */

        unsigned int            b;              /*     4     4 */
        unsigned char           c;              /*     8     1 */

        /* size: 12, cachelines: 1, members: 3 */
        /* sum members: 6, holes: 1, sum holes: 3 */
        /* padding: 3 */
        /* last cacheline: 12 bytes */
};
```

LSE Security System

Packed structs

```
struct foo_packed {
        unsigned char           a;              /*     0     1 */
        unsigned int            b;              /*     1     4 */
        unsigned char           c;              /*     5     1 */

        /* size: 6, cachelines: 1, members: 3 */
        /* last cacheline: 6 bytes */
} __attribute__ ((packed));
```

LSE Security System