x86a

Gabriel Laskar <gabriel@lse.epita.fr>

http://lse.epita.fr/teaching/epita/x86a.html

LSE Security System

Outline

- x86 assembly
- 64bit support
- pagination
- multi-core

LSE Security System

x86_64 : what's new ?

- more registers
- 64bit addresses, 64bit registers
- no more segmentation (but gdt still present)
- new features in pagination
- no Task Switch but TSS still present
- lots of thing removed, but still present (for special cases)

LSE Security System

Multiple kind of x86 registers

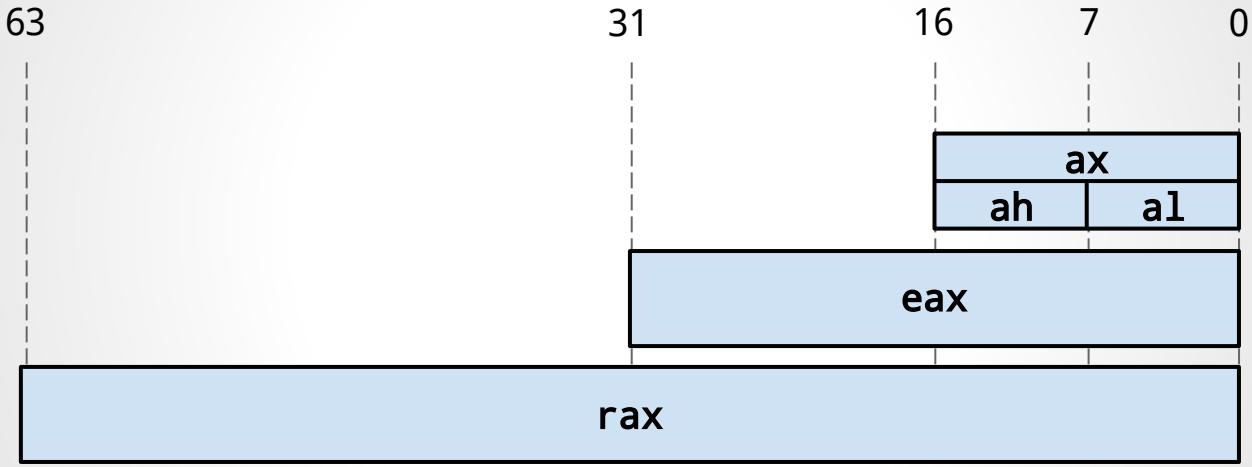
- General purpose registers
- Segment registers
- FLAGS
- Control & Memory registers

General purpose registers

- %rax, %rbx, %rcx, %rdx
- %rsi, %rdi
- %rsp, %rbp
- %rip
- %r8 → %r15

LSE Security System

Register Aliases



Instruction pointer : %rip

- in x86_64, instructions can now reference data relative to %rip

```
.global main
main:
    lea string(%rip), %rdi
    call puts
    ret

.section .rodata
string:
    .ascii "hello world!"
```

String manipulation

- rep prefix allow to repeat an instruction
- string instructions : movs, scas, stos

```
.global strlen
strlen:
    xor %rcx, %rcx
    not %rcx
    xor %al, %al
    cld
    repne scasb
    not %rcx
    dec %rcx
    mov %rcx, %rax
    ret
```

LSE Security System

Flags register

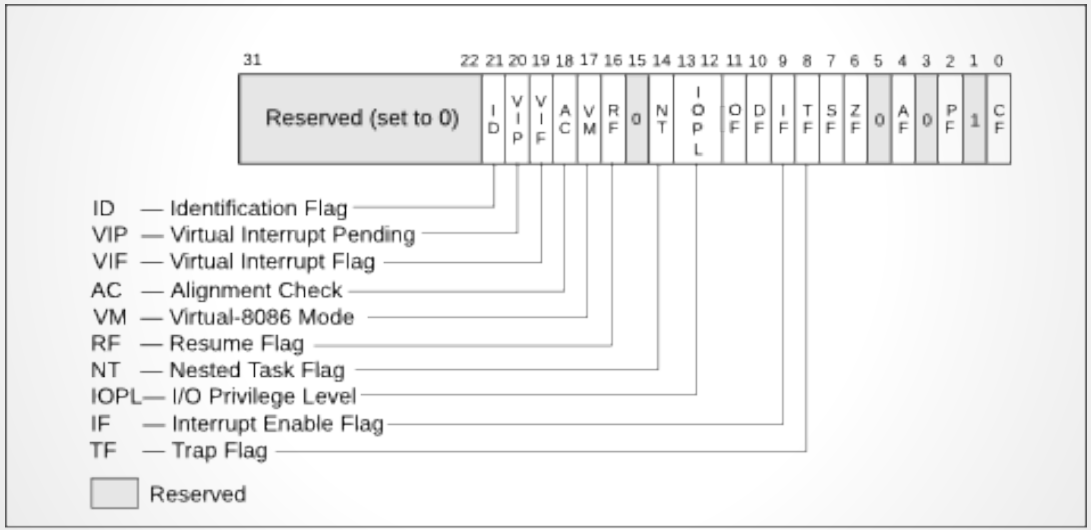


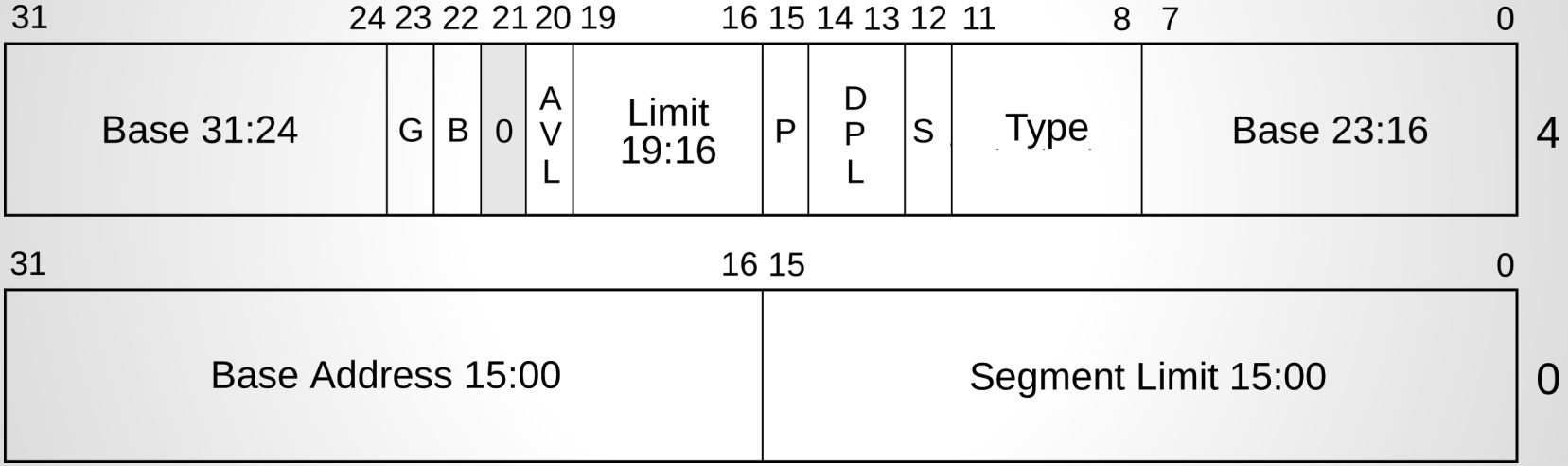
Figure 2-5. System Flags in the EFLAGS Register

Rings

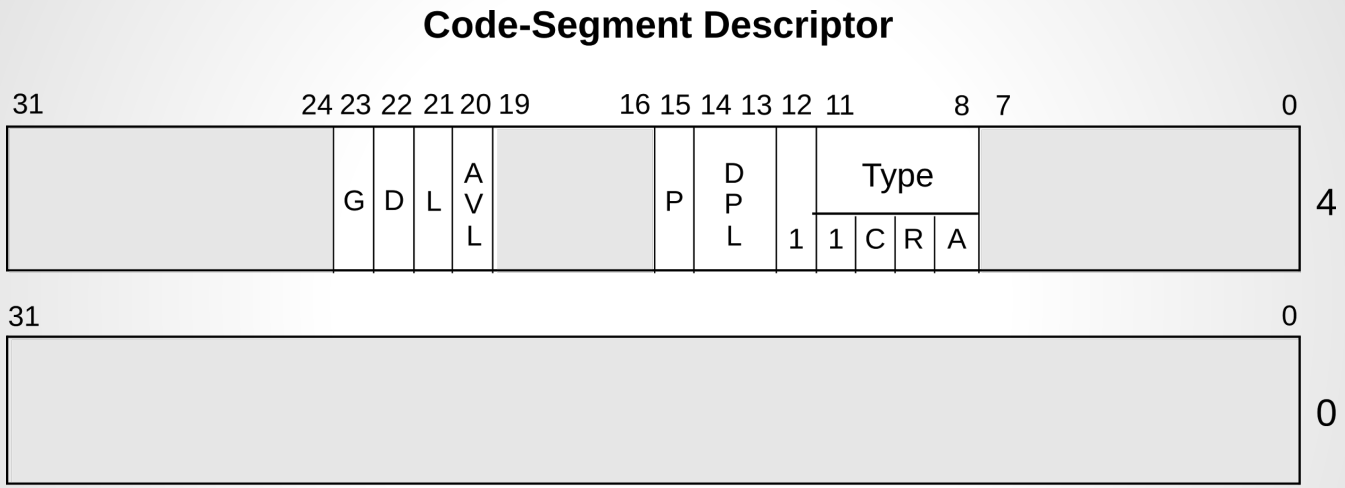
- 4 rings in x86_32, only 2 rings in x86_64
- SMM mode
- other modes (virtualization)

LSE Security System

GDT entries


31
24
23
22
21
20
19
16
15
14
13
12
11
8
7
0
Base 31:24
G
B
0
AVL
Limit 19:16
P
DPL
S
Type
Base 23:16
4
31
16
15
0
Base Address 15:00
Segment Limit 15:00
0

GDT entries in x86_64



A	Accessed
AVL	Available to Sys. Programmer's
C	Conforming
D	Default
DPL	Descriptor Privilege Level
L	64-Bit Flag
G	Granularity
R	Readable
P	Present

Segment selectors

- Tied to GDT entries
- 2 parts, public part and shadowed part
- provide basic permissions on zones
- each segment selector describe memory access for some instructions

LSE Security System

Descriptions of segment selectors

- cs : access to code (%rip, call, ret ...)
- ss : access to stack data (%rsp, push, pop)
- ds : access to memory and %rdi
- es : access to %rsi
- fs : user-defined
- gs : user-defined

LSE Security System

Thread local storage

- `%fs`, `%gs` can be used to implement TLS variables.
- One page mapped, and referenced by segment selector

Control registers

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

LSE Security System

Control Registers

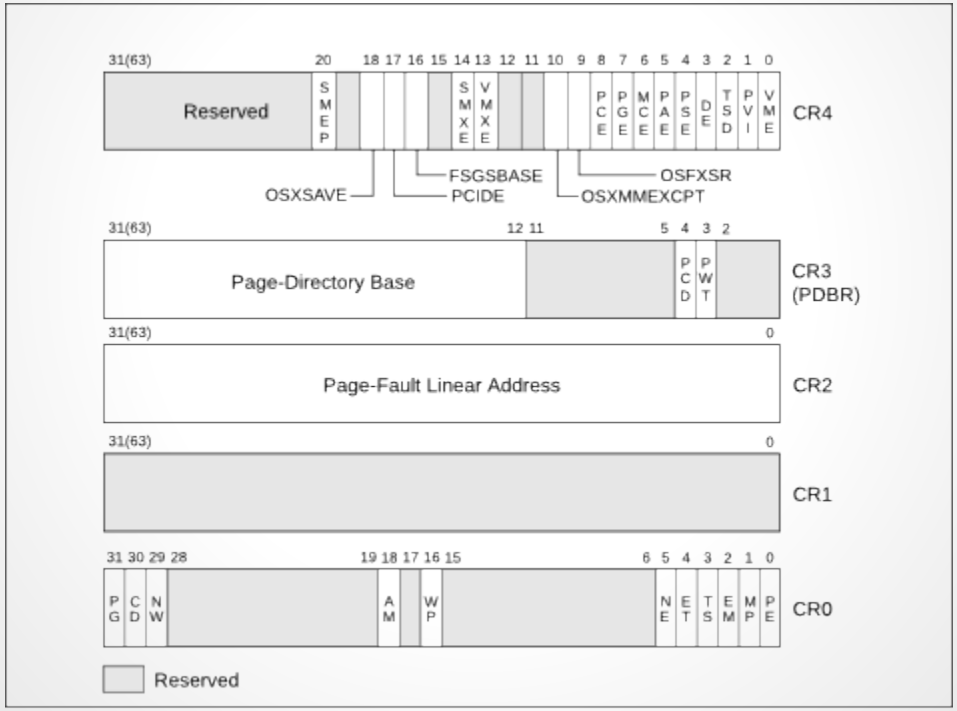


Figure 2-7. Control Registers

Debug registers

- support for debugging
- exceptions
- eflags register
- debug registers (`%dr0`-`%dr3`, `%dr6`, `%dr7`)

LSE Security System

Machine Specific registers

- Used to configure the internal state of the cpu
- accessed through 2 instructions:
  - rdmsr
  - wrmsr
- address specified in %ecx, and value in %edx:%eax

What can I do with MSRs?

- sysenter
- microcode updates
- mtrrs configuration
- smm configuration
- performance events & counters
- debug control
- misc features

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

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

LSE Security System

x86_64 : calling functions

- If the class is MEMORY, pass the argument on the stack.
- If the class is INTEGER, the next available register of the sequence %rdi, %rsi, %rdx, % rcx, %r8 and %r9 is used

LSE Security System

x86_64 : syscalls

- %rdi, %rsi, %rdx, %r10, %r8 and %r9
- The kernel destroys registers %rcx and %r11.
- instruction syscall
- The number of the syscall has to be passed in register %rax
- %rax contains the result of the system-call

Pagination

- multiple modes (32bit, 32bit pae, 64bit)
- table format
- TLB
- mirroring
- permissions
- initialization
- COW, swaping, shared memory

LSE Security System

Pagination

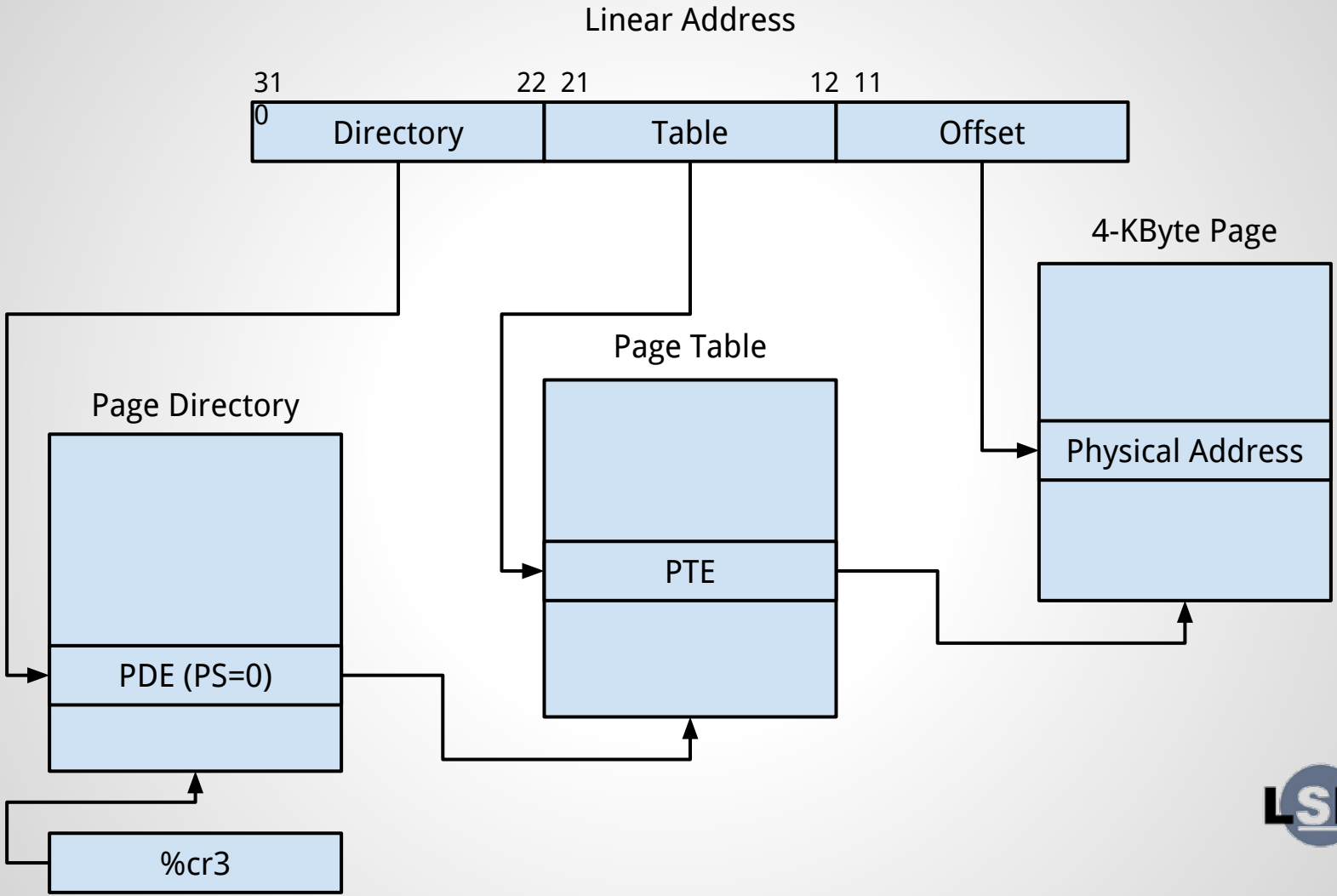
Table 4-1. Properties of Different Paging Modes

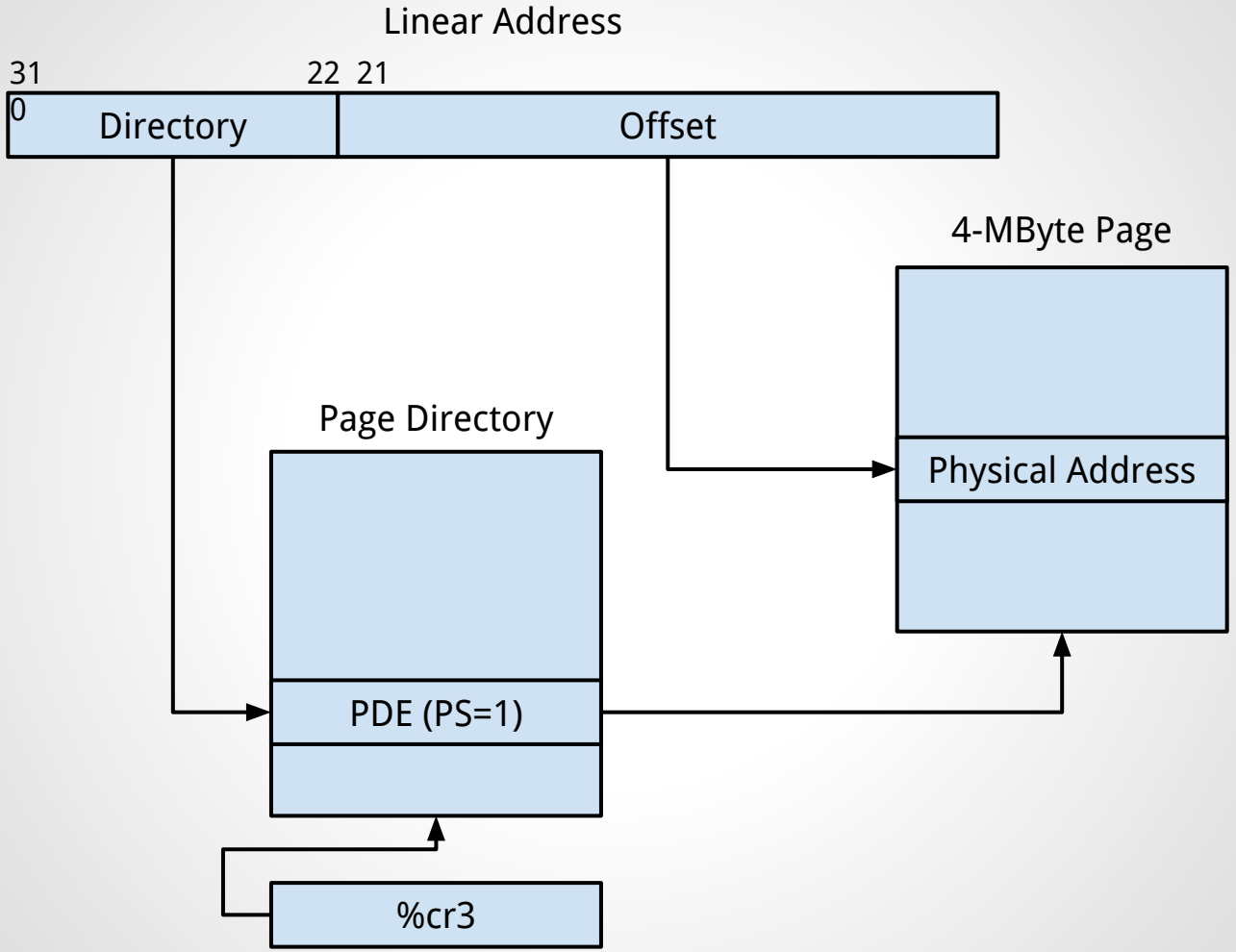
Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	Lin.-Addr. Width	Phys.-Addr. Width[1]	Page Sizes	Supports Execute-Disable?	Supports PCIDs?
None	0	N/A	N/A	32	32	N/A	No	No
32-bit	1	0	0[2]	32	Up to 40[3]	4 KB 4 MB[4]	No	No
PAE	1	1	0	32	Up to 52	4 KB 2 MB	Yes[5]	No
IA-32e	1	1	1	48	Up to 52	4 KB 2 MB 1 GB[6]	Yes[5]	Yes[7]

NOTES:

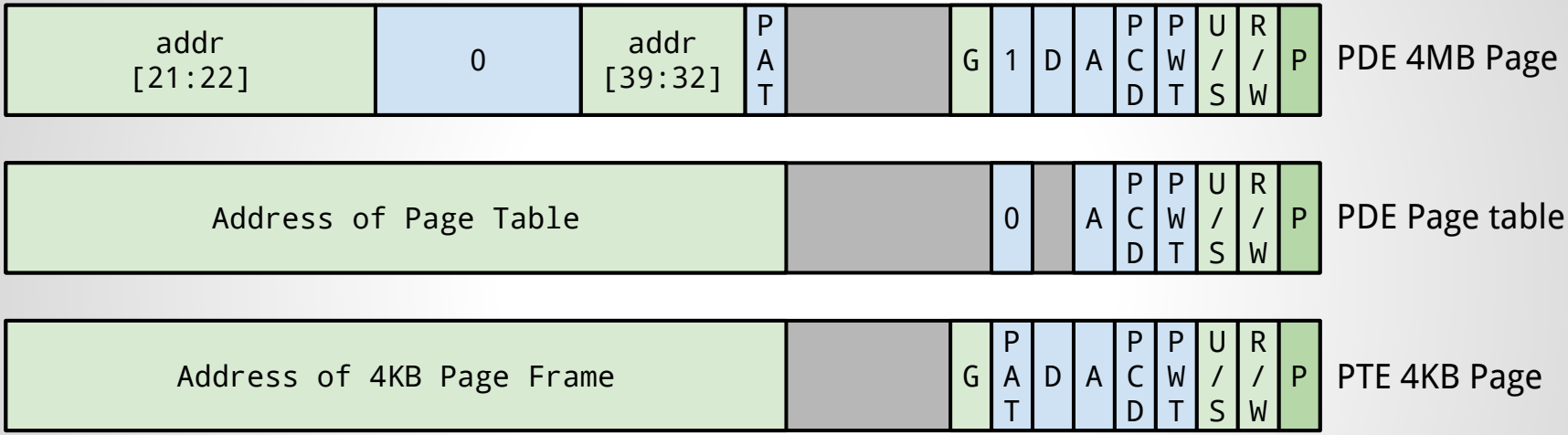
1. The physical-address width is always bounded by MAXPHYADDR; see Section 4.1.4.
2. The processor ensures that IA32_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.
3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 4.1.4 and Section 4.3.
4. 4-MByte pages are used with 32-bit paging only if CR4.PSE = 1; see Section 4.3.
5. Execute-disable access rights are applied only if IA32_EFER.NXE = 1; see Section 4.6.
6. Not all processors that support IA-32e paging support 1-GByte pages; see Section 4.1.4.
7. PCIDs are used only if CR4.PCIDE = 1; see Section 4.10.1.

LSE Security System


Linear Address
31
22 21
12 11
0
Directory
Table
Offset
4-KByte Page
Physical Address
Page Table
PTE
Page Directory
PDE (PS=0)
%cr3

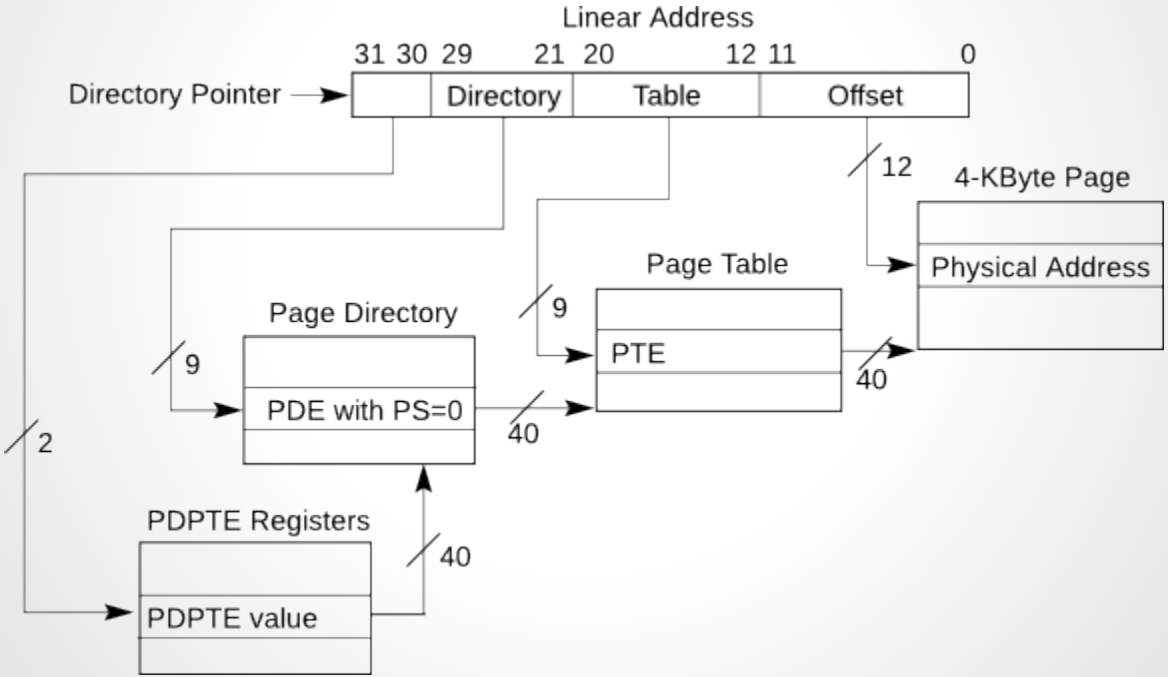

Linear Address
31
22
21
0
Directory
Offset
4-MByte Page
Physical Address
Page Directory
PDE (PS=1)
%cr3

PDE and PTE

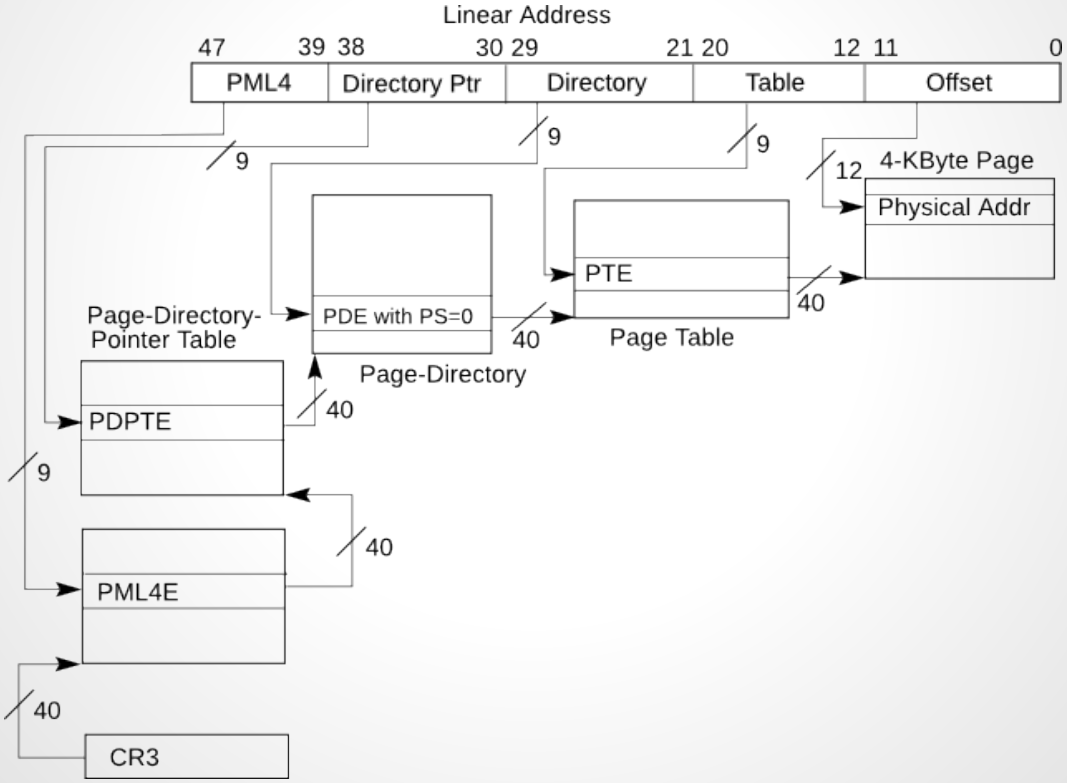


- R/W: Read/Write
- U/S: User/System
- PWT: Page Level write-through
- PCD: Page Level Cache disable
- A: Accessed
- D: Dirty
- G: Global (if `%cr4.pge = 1`)
- PAT: Reserved

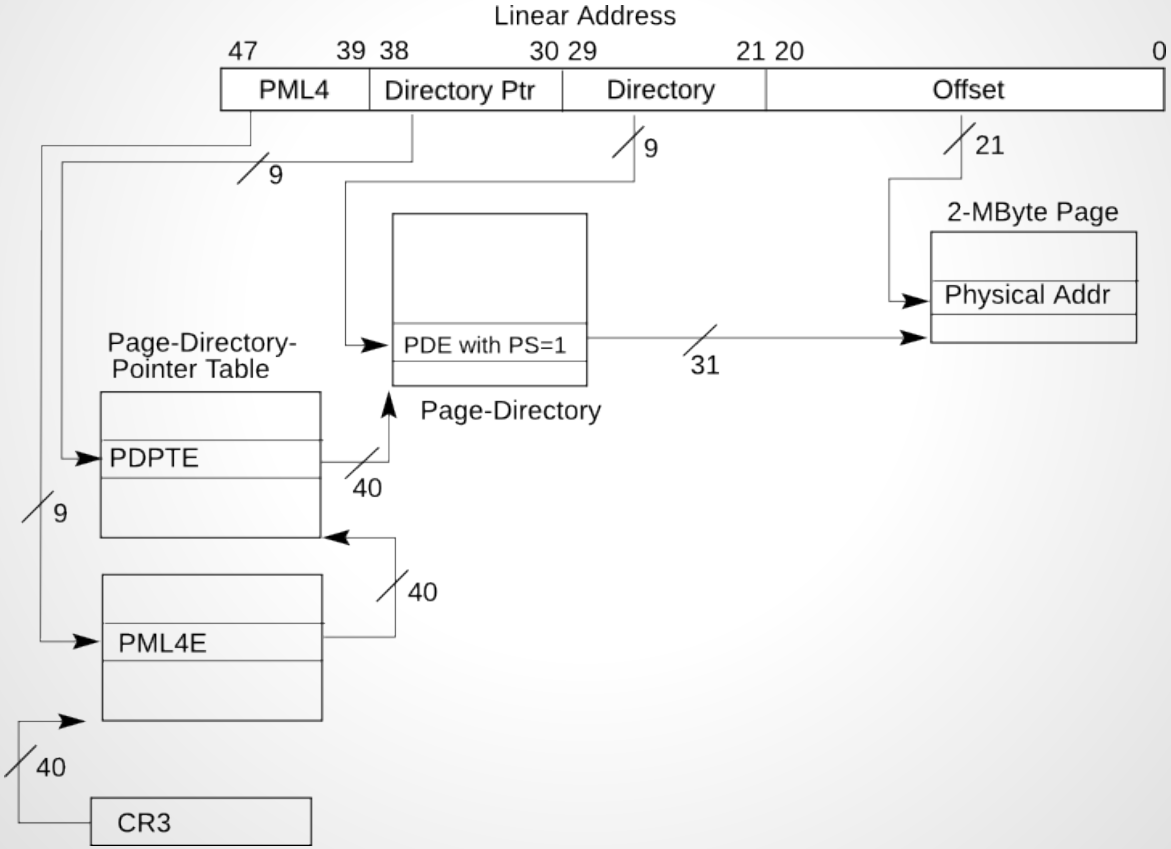
PAE


Linear Address
31 30 29 21 20 12 11 0
Directory Pointer
Directory
Table
Offset
12
4-KByte Page
Physical Address
Page Table
9
PTE
40
Page Directory
9
PDE with PS=0
40
2
PDPTE Registers
40
PDPTE value

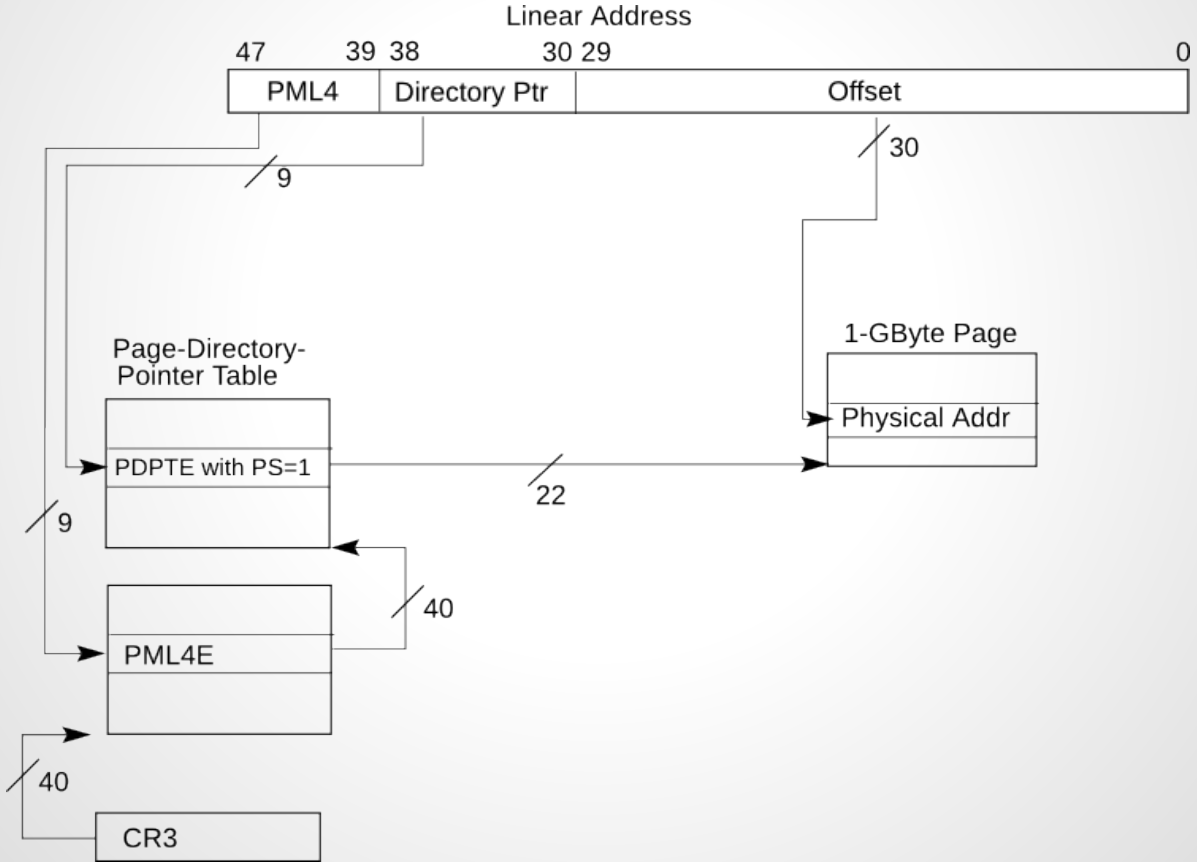
64bit pagination



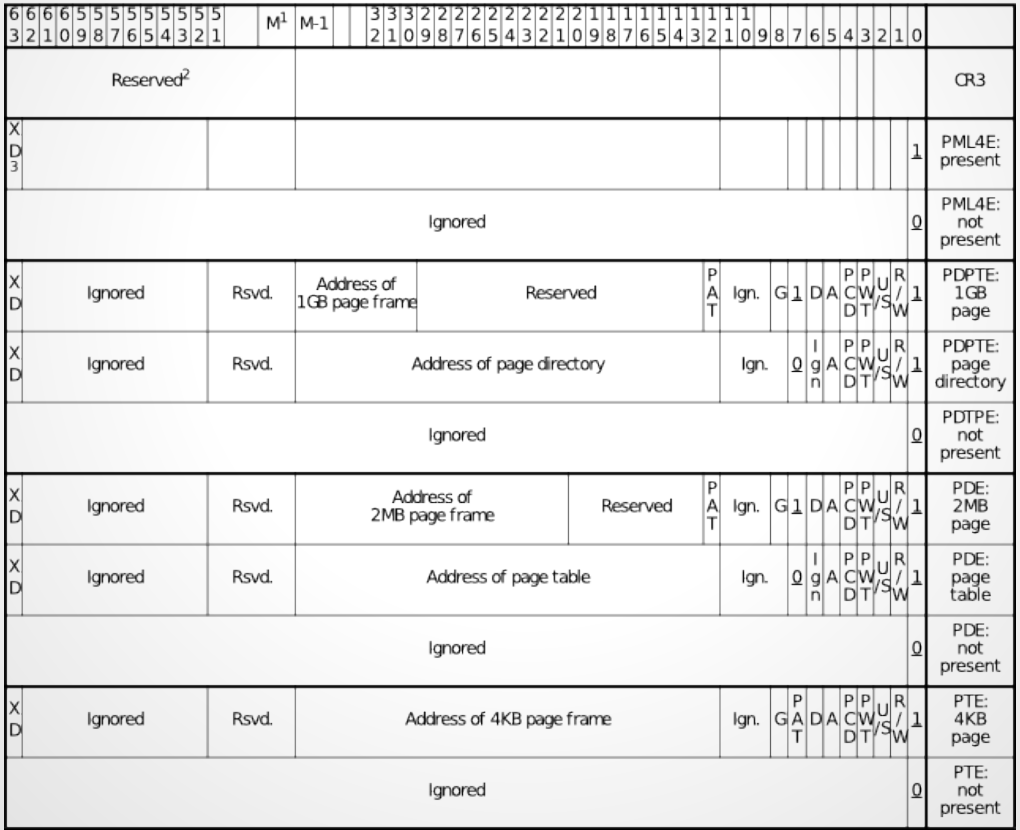
x86_64 : 2Mb Pages



x86_64 : 1Gb pages



x86_64 : structures



Page Fault Handling

- Which address? Content of %cr2
- Error Code:
  - P: non-present (clear), page-level protection violation (set)
  - W/R: read (clear) or write (set) error
  - U/S: supervisor (clear) or user-mode (set)
  - RSVD: reserved bit violation (set)
  - I/D: data (clear) or instruction (set)

TLB

- Cache for address translations
- 2 TLB : one for data, one for instructions

LSE Security System

PAX

On x86_32, How can we enforce NX bit without the hardware support ?

LSE Security System

Multi Core

- bsp/ap initialization
- mptables, madt
- idt, ipi, lapic, ioapic
- impact on kernel code
- Kernel Lock
- cache coherency

LSE Security System

x86_64 Initialization

- Disable paging
- Set the PAE enable bit in `%cr4`
- Load `%cr3` with the physical address of the PML4
- Enable long mode by setting the `EFER.LME` flag in MSR `0xC0000080`
- Enable paging

LSE Security System

x86_64 : Are we done yet ?

- We are still in compatibility mode, with 32-bit code
  - reload segment selector for %cs with
    - DB = 0
    - L = 1
- Now we can relocate all other tables (idt, gdt, tss...)

LSE Security System

Interrupt Routing

- If I have multiple core, to which core the interrupt are delivered ?
- We need a new mechanism that enable customisation for interrupt routing

LSE Security System

LAPIC

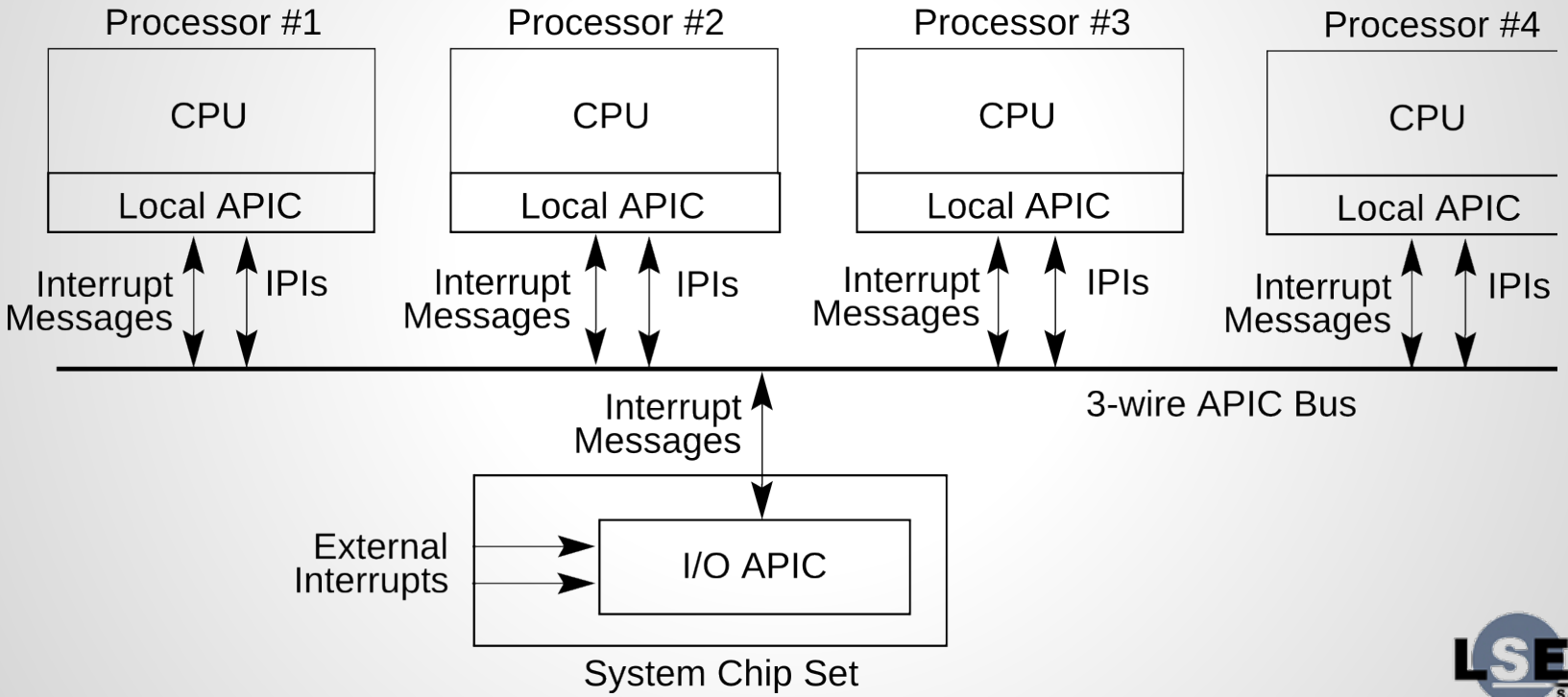
- memory mapped (starting at `0xfee00000`)
- Receive interrupts from multiple sources
  - Locally connected I/O devices (Local & External)
  - Inter-processor interrupts (IPIs)
  - APIC timer, PMC, Thermal, internal errors

LSE Security System

IOAPIC

- 83093AA
- at least 24 programmable interrupts
- memory mapped
- more flexible on priorities
- usually connected to the LAPICs

IRQ Routing


Processor #1
Processor #2
Processor #3
Processor #4
CPU
CPU
CPU
CPU
Local APIC
Local APIC
Local APIC
Local APIC
Interrupt Messages
IPIs
Interrupt Messages
IPIs
Interrupt Messages
IPIs
Interrupt Messages
IPIs
3-wire APIC Bus
Interrupt Messages
External Interrupts
I/O APIC
System Chip Set

Talking to another core : IPI

- In the LAPIC
- can send unicast or broadcast requests
- Used for :
  - flushing TLBs
  - flushing Caches
  - power up or down another core
  - arbitrary messages

LSE Security System

Caching

- Caches are either shared (L2)
- or specific for a core (L1)
- Synchronisation must be done at the hardware level

Discover Multiple cores

- How many cores do I have ?
- Where is located my APICs ?
- How the interrupt are configured ?

Multiprocessor Specification

- Old deprecated interface
- Easy to use
- But first we must find it !

Where are my MP tables

- Find the MP Floating Pointer Structure
  - In the first kilobyte of the EBDA
  - In the first kilobyte of system base memory (639k → 640k, or 511k → 512k)
  - In the BIOS ROM address space `0xf0000` and `0xffffff`
- Search for the Magic Value "_MP_"

What’s in it ?

- Processor
- Bus (PCI, ISA, VESA, etc...)
- I/O APIC configurations
- I/O Interrupts assignment
- Local Interrupts assignment

ACPI

- provides an open standard for device configuration and power management
- Replace
  - Advanced Power Management
  - MultiProcessor Specification
  - Plug and Play BIOS Specification

ACPI Tables

- Root System Description Pointer (RSDP)
- System Description Table Header
- Root System Description Table (RSDT)
- Fixed ACPI Description Table (FADT)
- Differentiated System Description Table (DSDT)
- Multiple APIC Description Table (MADT)
- Extended System Description Table (XSDT)
- ...

Root System Description Pointer

- Contains address of RSDT and XSDT
- Still in placed at random point in memory
- Magic "RSD PTR "

Root System Description Table

- Header with information about vendor
- Contain addresses to other tables
- XSDT is the same table but with 64-bit addresses

Fixed ACPI Description Table

- Define ACPI information vital to an ACPI-compatible OS
- Registers
- Pointer to DSDT
- Contains also various information (how to enable or disable ACPI)

Differentiated System Description Table

- Contains AML Code blocks
- AML is a generic bytecode
- Describe Hardware configuration
- Contains calls for Power Management states

Multiple APIC Description Table

- APIC structures
- Processor descriptions

LSE Security System

Multi Core initialization

- Parse the MP tables to find the other APICs.
- initializes the bootstrap processor's local APIC.
- send Startup IPIs to each other cores with the address of trampoline code.
- trampoline code initializes the AP's to protected mode
- The BSP can initialize the IO APIC into Symmetric IO mode, to allow the AP's to begin to handle interrupts.
- The OS continues further initialization, using locking primitives as necessary.

LSE Security System

Changes in the OS

- kind of like multi-threaded application
- We need to care about locking
- And never stop the other cores

LSE Security System

Per-cpu context

- Per-cpu context
  - Most of the control structures are per-cpu
  - Some can be shared, for example GDT
- Per-cpu variables
  - we can use %gs or %fs to implement per-cpu pages.

Changes in the OS

- Locking strategies
  - Giant Lock (Big Kernel Lock)
  - Fine grained lock
- Algorithms
  - Scheduling
  - Memory allocation
  - Handling of kernel resources

LSE Security System