reports

本报告已设置目录

1. x86 系统架构概览

x86是一种广泛使用的指令集架构（ISA），由 Intel 最初在 1978 年发布的 16 位处理器 8086 引入。x86 架构经历了多次演进，从 16 位扩展到 32 位（x86——32），再发展到 64 位（x86-64，也叫 AMD 64），如今应用于台式机、笔记本电脑、服务器等多种设备。

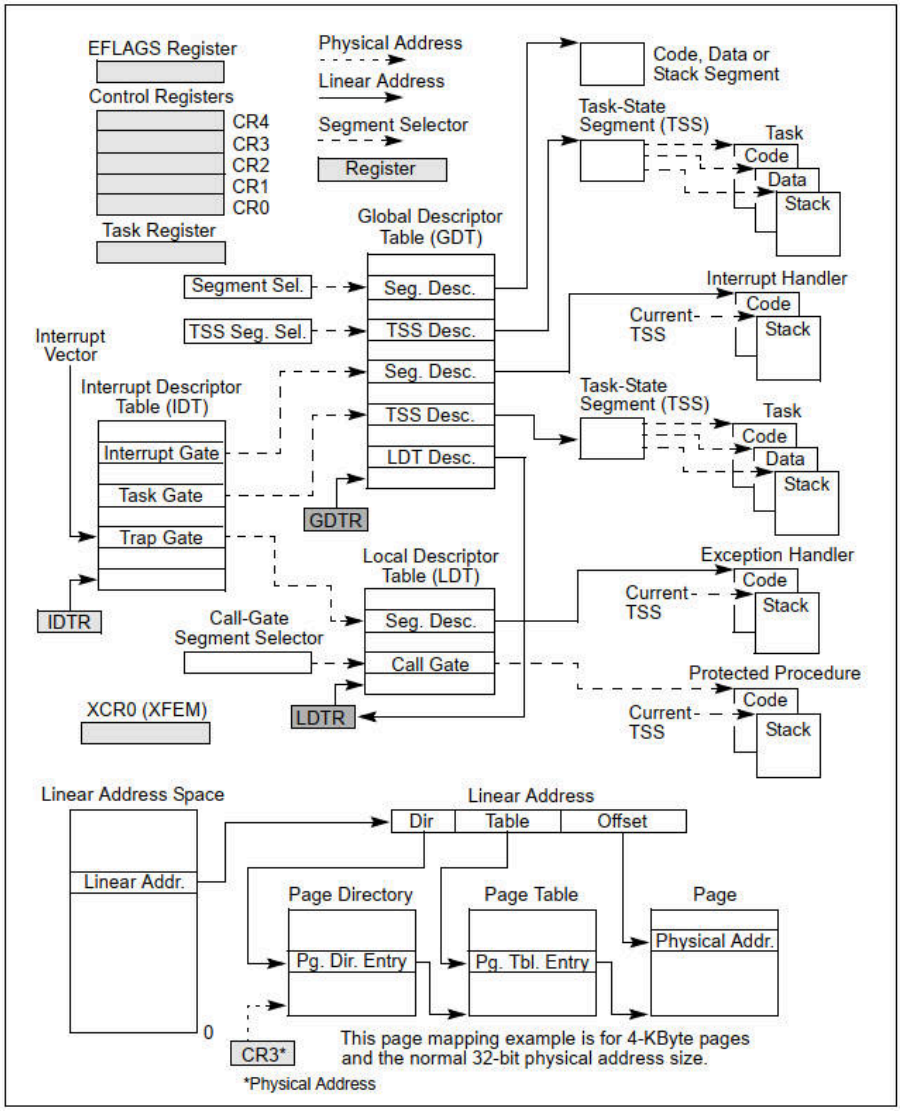
笔记的内容主要基于 IA-32 架构给出 x86 系统体系结构的内容阅读以及理解。

IA-32 架构为操作系统和系统软件开发提供了广泛的支持，提供了多种操作模式：实模式、保护模式、虚拟 8086 模式和系统管理模式。

1.1 系统级体系结构概览

系统级架构由一组寄存器、数据结构和指令组成，旨在支持基本的系统级操作，如内存管理、中断和异常处理、任务管理和多处理器控制。

下图展示了 IA-32 架构系统级的寄存器和数据结构



接下来是关于其中一些重点内容的介绍。

- 段：*段是一种逻辑分区，它允许程序将数据和代码划分为不同的区块，例如代码段、数据段和堆栈段。*
- 段描述符：负责提供段的基址，访问权限、类型和该段的用途等信息。

Global and Local Descriptor Tables（全局和局部描述符表）

在保护模式下运行时，所有的内存访问都会通过全局描述符表（GDT）或可选的局部描述符表（LDT），如下图所示。这些表包含了称为段描述符(Seg. Desc.）的条目。段描述符提供了段的基地址以及访问权限、类型和使用信息。

> 传统的实地址模式下的汇编中，利用 CS:IP 来访问内存，CS 左移四位加 IP 获得物理地址，但是通常不需要关心基地址，只需要偏移地址即可访问内存，因为实地址模式的基地址通常是固定的。但是在保护模式下，为了达到保护内存的目的，就需要让不同程序有不同的基地址（基地址如果固定，那么很容易就被他人直到程序的运行位置，很不安全）。所以，在保护模式下，内存是一段一段分

> 配的，分配的那一段内存头地址就是基地址，也称段地址。同时，操作系统也需要记录分配出去的每一段内存，实际上，每段内存都使用一个 8 字节的段描述符记录着，所有的这些段描述符被放在一块连续内存空间中存储着，称为描述符表。

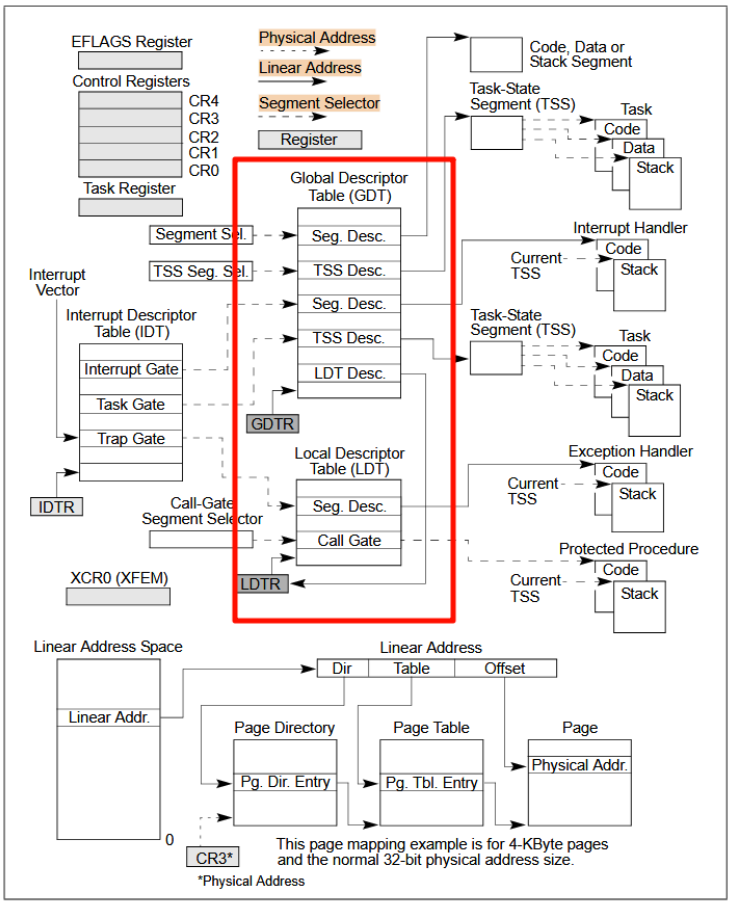


Figure 2-1. IA-32 System-Level Registers and Data Structures

每个段描述符都有一个相关的段选择器。段选择器为使用它的软件*提供了一个索引到 GDT 或 LDT*（其相关联的段描述符的偏移量）、*一个全局/局部标志*（决定选择器指向 GDT 还是 LDT），以及*访问权限信息*。要访问段中的一个字节，*必须提供段选择器和偏移量*。段选择器提供了对段的段描述符的访问（在 GDT 或 LDT 中）。从段描述符中，处理器获得了*线性地址空间中段的基地址*。*然后偏移量提供了相对于基地址的字节位置*。只要当前处理器运行的特权级别（CPL）可以访问该段，这种机制可以用来访问任何有效的代码、数据或栈段。CPL 被定义为当前执行的代码段的保护级别。

见图 2-1。图中的实线箭头表示线性地址，虚线表示段选择器，点线箭头表示物理地址。为了简化，许多段选择器被显示为直接指向一个段的指针。然而，从段选择器到其相关联的段的实际路径始终是通过 GDT 或 LDT。

GDT 的基地址的线性地址包含在 GDT 寄存器（GDTR）中；LDT 的线性地址包含在 LDT 寄存器（LDTR）中。

1. GDT (Global Descriptor Table) - 全局描述符表

定义与作用

- GDT 是一个系统范围内的表，用于保存所有任务和进程共享的段描述符。
- 每个描述符定义了一个段的 基址（Base Address）、段限长（Limit）、以及访问权限（Access Privileges）。
- 主要用于描述代码段、数据段、系统段（如任务状态段 TSS）。

GDT 结构

每个 GDT 描述符长度为 8 字节，其字段如下：

- Base Address (32 bits)：段的基地址。
- Limit (20 bits)：段的大小（最大可达 4 GB）。
- Access Byte (8 bits)：控制段的类型（如代码段/数据段）和访问权限。
- Flags (4 bits)：指定段的粒度和默认操作大小（16/32 位）。

GDT 的作用示例

- 定义内核模式和用户模式的代码段、数据段。
- 定义系统段，如 TSS，用于任务切换。

2. LDT (Local Descriptor Table) - 本地描述符表

定义与作用

- LDT 是与特定任务或进程相关的段描述符表，每个进程可以拥有自己的 LDT。
- 它允许不同任务或进程拥有各自的虚拟地址空间和内存段定义。
- LDT 中的段描述符结构与 GDT 中的类似。

LDT 的用途

- 为不同任务或进程定义私有数据段和代码段，实现进程的隔离。
- 在进程上下文切换时切换到对应的 LDT。

3. GDT 与 LDT 的区别

特性	GDT (Global Descriptor Table)	LDT (Local Descriptor Table)
作用范围	全局系统范围，所有进程共享	进程/任务级别，每个任务可有独立的 LDT
主要用途	描述系统段（如内核代码段、TSS）	描述进程特定的代码段和数据段
描述符类型	包含代码段、数据段、TSS 等	包含进程私有的段描述符
表的数量	系统中只有一个 GDT	每个任务或进程可以有一个 LDT

System Segments, Segment Descriptors, and Gates (系统段，段描述符，门)

系统段

- 系统段：除了代码、数据、堆栈段，其组成了一个程序的运行环境外，该体系结构还定义了两个系统段：`任务状态段` (the task-state segment—TSS) 和 `LDT`。GDT 因其不能通过段选择子 - 段描述符的方式访问，从而不被看作是一个段。TSSs 和 LDTs 都有为其定义的段描述符。

> 之所以 GDT 不能被视作段，是因为 GDT 不作为段选择子的最终目的，段选择子需要通过段描述符进行索引到具体的段，而 LDT 可以通过 GDT 中的 LDT Desc. 来被视作一个具体的段进行使用。

段描述符

- 段描述符：负责提供段的基址，访问权限、类型和该段的用途等信息。

门

门 (Gate) 机制

x86 架构定义了几种特殊的描述符，称为门 (Gate)，用于在不同特权级之间创建受保护的访问路径。门包括：

- 调用门 (Call Gate)：用于跨特权级调用其他代码段中的函数。
- 中断门 (Interrupt Gate)：处理硬件中断。
- 陷阱门 (Trap Gate)：处理软件中断和异常。
- 任务门 (Task Gate)：用于任务切换。

这些门提供了在不同特权级（Privilege Levels，PL）之间安全地进行跳转的途径。

特权级检查与门的作用

x86 体系中有 4 个特权级，编号为 0（最高级）到 3（最低级）。通常，操作系统内核运行在级别 0，用户程序运行在级别 3。门控制跨级访问时的安全性：

- CALL 指令调用门（Call Gate）：
  - 访问一个更高特权级（数值更低）的代码段，需要通过 调用门。
  - 当调用方提供了调用门的段选择子，CPU 会检查当前特权级（CPL，Current Privilege Level）与目标代码段的权限。

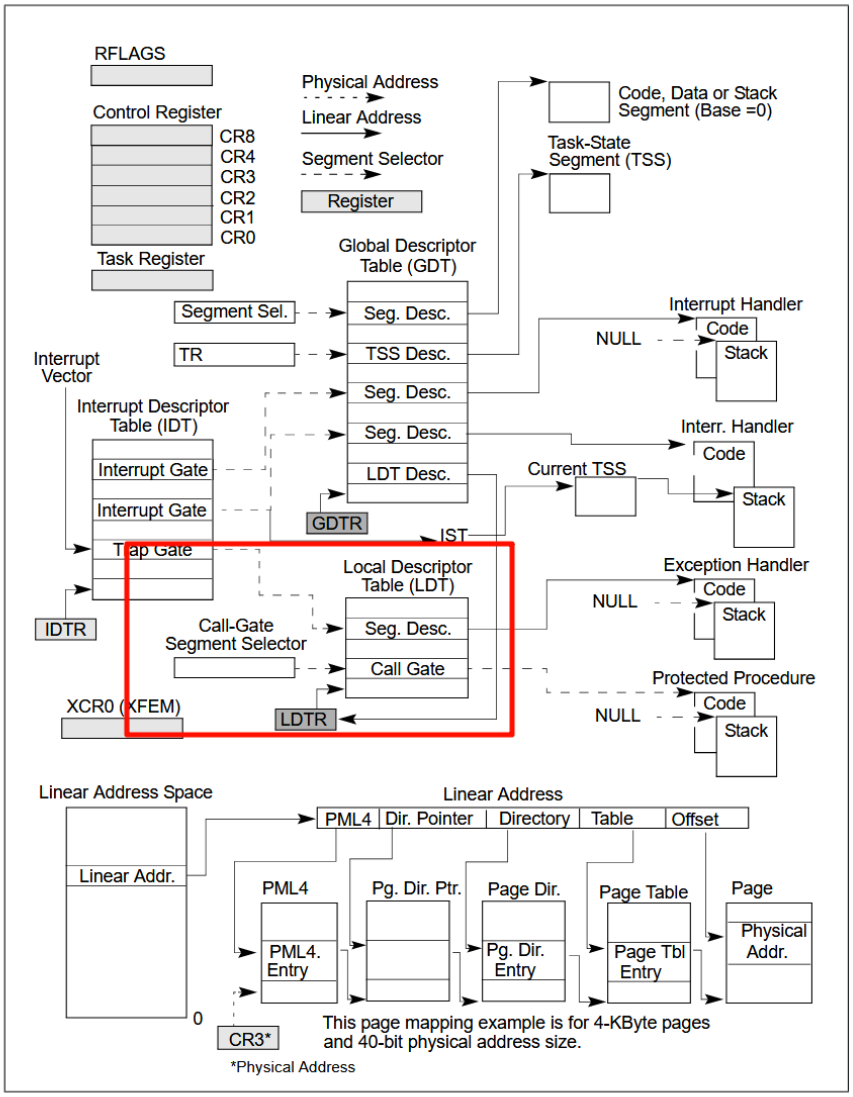


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode

访问检查的过程

1. CPU 检查 CPL 与 调用门和目标代码段的权限：
   - 如果权限允许，CPU 会从调用门中获取目标代码段的选择子和偏移地址。

- 特权级提升时，CPU 会切换到更高特权级的堆栈。

2. 堆栈切换：
    - 如果调用需要从用户模式（级别 3）切换到内核模式（级别 0），CPU 会从 TSS 获取新的堆栈指针，并切换到新堆栈。

门与代码段长度的转换

- 门还支持 16 位与 32 位代码段之间的切换。
    - 这在系统中非常重要，因为某些设备驱动程序或旧代码可能仍使用 16 位代码段，而现代系统则运行 32 位或 64 位代码。

Task-State Segments and Task Gates（任务状态段与任务门）

1. TSS（Task-State Segment）是什么？

- TSS 是一个特殊的系统段，用于保存一个任务的执行状态。
- TSS 存储的内容：
    - 通用寄存器（如 `EAX`，`EBX`，`ECX` 等）。
    - 段寄存器（如 `CS`，`DS`，`SS` 等）。
    - EFLAGS 寄存器：存储条件标志位，用于控制程序执行。
    - EIP 寄存器：存储下一条要执行的指令地址。
    - 堆栈指针和段选择子：为每个特权级（Ring 0、1、2）保存一个单独的堆栈段选择子和指针。
    - LDT 选择子：指向当前任务所使用的 LDT。
    - CR3 控制寄存器：存储 分页表的基地址，用于虚拟内存管理。

TSS 的作用

- 当切换任务时，当前任务的状态被存储到 TSS，以便稍后可以恢复。
- 新任务的状态会从它的 TSS 中加载到 CPU 的各类寄存器中。

2. 任务切换的过程

当任务切换时，CPU 会执行一系列操作，将执行环境从当前任务切换到新任务：

任务切换步骤

1. 存储当前任务的状态：

- CPU 将当前任务的状态保存到当前 TSS，包括寄存器、标志位、段选择子、堆栈指针等。

2. 加载新任务的 TSS 段选择子：
   - 将 新任务的 TSS 选择子加载到 任务寄存器（Task Register，TR）中。
3. 通过 GDT 访问新任务的 TSS：
   - CPU 在 GDT 中找到新任务的 TSS 段描述符，并访问新任务的 TSS。
4. 恢复新任务的状态：
   - CPU 将 新任务的状态从 TSS 中加载到所有相关寄存器（包括通用寄存器、段寄存器、CR3 寄存器、EFLAGS、EIP 等）。
5. 开始执行新任务：
   - CPU 跳转到 新任务的 EIP 地址，从该地址开始执行新任务的代码。

3. 任务门（Task Gate）与调用门的区别

- 任务门（Task Gate） 和 调用门（Call Gate） 都是特殊的段描述符，用于跨特权级跳转，但它们的功能有所不同：

特性	任务门（Task Gate）	调用门（Call Gate）
指向的目标	指向TSS（任务状态段）	指向代码段
作用	切换到新任务	调用另一个特权级的函数
切换的状态	保存当前任务并加载新任务	保持当前任务不变
堆栈切换	切换到新任务的堆栈	可能根据特权级切换堆栈

任务门的用法

- 任务门提供了一种受保护的途径，允许系统通过指定的入口点访问一个新任务。
- 使用任务门的 CALL 或 JMP 指令会引发任务切换，并加载新任务的 TSS。

Interrupt and Exception Handling（中断和异常处理）

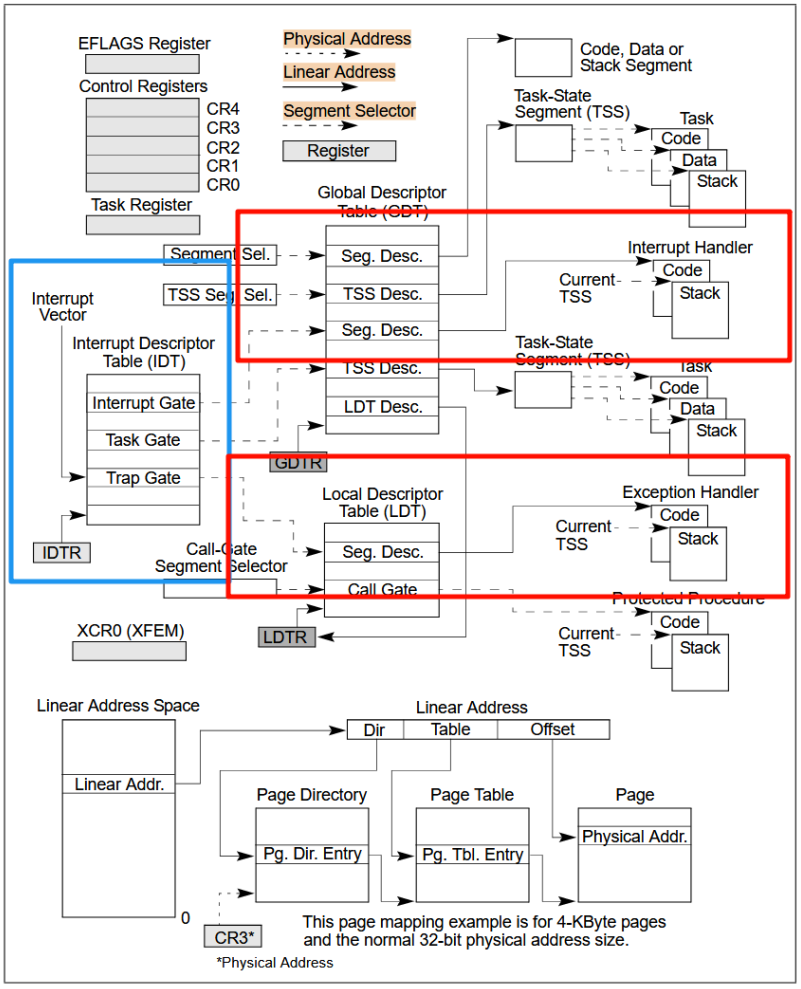


Figure 2-1. IA-32 System-Level Registers and Data Structures

IDT（Interrupt Descriptor Table，中断描述符表）

- IDT 是一个用于存储中断和异常处理程序入口的表。
- 每个 IDT 条目 是一个门描述符（Gate Descriptor），指向特定的中断或异常处理程序。
- 和 GDT 一样，IDT 不是一个段。它的基地址保存在 IDTR（IDT 寄存器）中，并通过该寄存器访问。

IDT 中的 Gate Descriptors（门描述符）

IDT 中的描述符可分为三种类型：

1. Interrupt Gate（中断门）：用于处理硬件中断。
2. Trap Gate（陷阱门）：用于处理软件中断或异常。
3. Task Gate（任务门）：用于在某些异常情况下触发任务切换。

中断处理流程

中断来源包括：

- 外部硬件中断（例如键盘、网络卡）。
- 内部异常（如除零错误、缺页错误）。
- 软件中断（由指令 `INT`、`INT3`、`INTO` 等触发）。

具体流程

1. CPU 接收到中断向量：
   - 中断向量（Interrupt Vector）是一个**编号**，由硬件控制器或软件传递给 CPU，用于**标识具体的中断或异常类型**。
2. **查找 IDT 中对应的描述符**：
   - 中断向量作为**索引**，在 IDT 中找到对应的 Gate Descriptor（**网关描述符**）。
3. **根据描述符类型处理中断**：
   - Interrupt Gate 或 Trap Gate：
     - 访问对应的**中断或异常处理程序**，*类似于*通过 **调用门**（Call Gate） 调用代码。
     - 在 Interrupt Gate 中，CPU 会**自动屏蔽进一步的中断**，防止嵌套中断。
     - Trap Gate 处理完后，CPU 不屏蔽中断，可以继续接收其他中断。
   - Task Gate：
     - 触发**任务切换**，CPU 会根据任务门访问一个新的 TSS（**任务状态段**），从而将控制权交给**另一个任务**来处理。

Interrupt Gate vs Trap Gate vs Task Gate

Gate 类型	用途	处理方式	特点
Interrupt Gate	处理硬件中断	屏蔽进一步的中断，执行完毕后恢复	防止嵌套中断
Trap Gate	处理异常或软件中断	不屏蔽其他中断，可以继续处理其他事件	常用于断点调试等情况
Task Gate	触发任务切换	切换到新的任务状态段（TSS），并恢复任务状态	用于复杂的异常处理，如缺页异常

中断处理例子

当一个 **键盘中断** 触发时，流程如下：

1. **中断控制器**发送一个中断向量给 CPU（假设编号为 33）。

2. CPU 在 IDT 中查找 第 33 个条目，找到一个 Interrupt Gate。
3. CPU 切换到内核模式并调用键盘中断处理程序。
4. 在处理完键盘输入后，CPU 恢复中断前的状态，继续执行原来的程序。

Memory Management（内存管理）

物理地址和虚拟地址

- **物理地址**：直接指向内存中的物理位置，每次访问的地址都是实际的物理内存地址。这种模式称为**直接物理寻址**，主要用于不支持虚拟内存的系统。
- **虚拟地址（Linear Address）**：
  - 在启用**分页机制**时，程序访问的地址不再是直接的物理地址，而是一个**线性地址（虚拟地址）**。
  - 线性地址通过**分页结构**映射到物理地址。
  - 分页允许系统使用**部分物理内存**存储当前最活跃的页面，并将不常 用的页面 **换出到磁盘**，实现内存的虚拟化。

分页结构工作原理

分页是将**虚拟地址空间**划分为固定大小的块（通常为 4KB），每个块称为**页面**（page）。分页结构负责将这些虚拟页面映射到**物理内存页框**（page frames）。

分页机制中的重要概念

- **分页结构（Paging Structures）**：
  - 分页结构存储在**物理内存**中，并保存虚拟页面与物理页框之间的映射关系。
  - 分页结构的基地址存储在控制寄存器 CR3 中。
- **CR3 寄存器**：
  - CR3 寄存器保存分页结构的**基地址**。
  - 在多任务系统中，每个任务或进程可以有**独立的分页结构**，以实现内存的隔离。

分页的优点

- **内存隔离**：
  - 每个任务或进程可以拥有**独立的分页结构**，避免进程之间的相互干扰。
- **换页机制（Page Swapping）**：
  - 只有**最近访问的页面**会保存在物理内存中，其他页面可以换出到磁盘，

节省物理内存。

- 内存保护：
  - 页表中的条目不仅存储物理地址，还包含访问权限（如只读、可写），防止非法访问。

GDT 和 IDT 分页

- GDT 和 IDT 的内容存储在物理内存中，也可以被虚拟化和分页，存入虚拟内存中。
  - GDT 的内容*可能位于多个分页中的不同位置*。
  - 如果 GDT 的某个部分被换出物理内存，当 CPU 需要访问它时，会触发缺页异常，系统会将对应页面重新加载。

分页结构层级

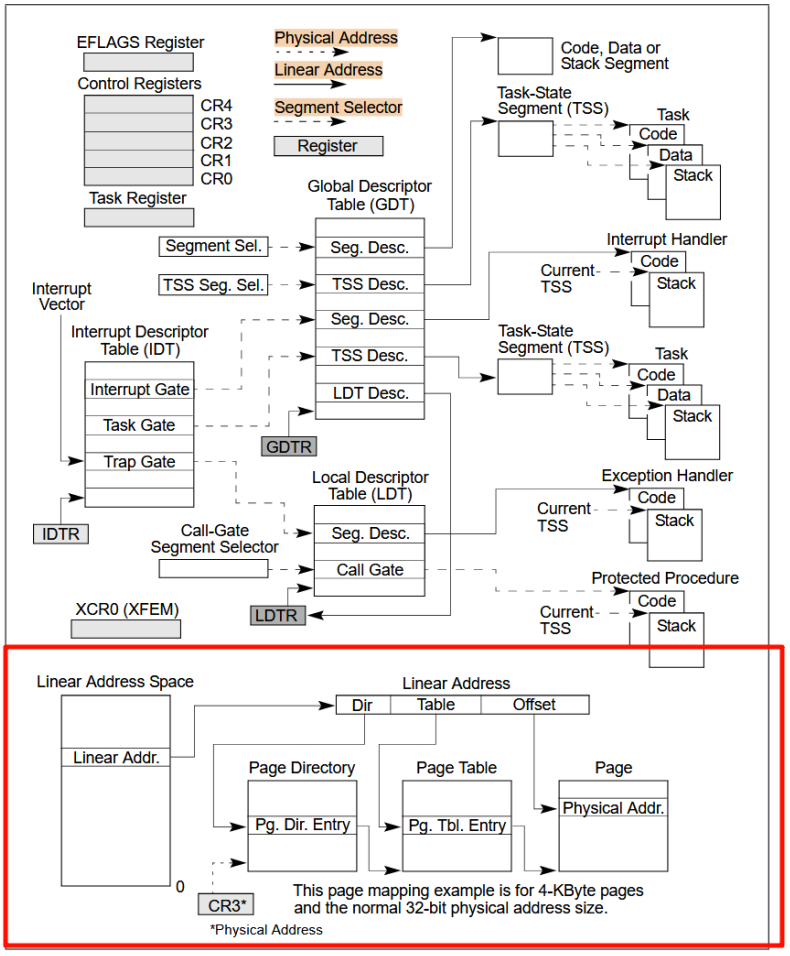


Figure 2-1. IA-32 System-Level Registers and Data Structures

32 位分页的层级（见 Figure 2-1）

在32 位分页模式下，分页结构通常包括以下多级页表：

1. 页目录表（Page Directory Table）：
    - CR3 寄存器指向的就是页目录表的基地址。
    - 每个*目录项*指向一个页表。
2. 页表（Page Table）：
    - 每个*页表项*指向一个**物理页框**，即物理内存中的实际存储位置。
3. 页面（Page Frame）：
    - 页框是实际的数据存储单元，大小通常为4KB。

虚地址解析

在分页机制中，**虚拟地址被拆分为多个部分，每一部分用于索引不同级别的分页结构**：

- **虚拟地址结构**：
    - **页目录索引**：用于在页目录表中定位目标页表。
    - **页表索引**：用于在页表中找到具体的页框。
    - **页内偏移（Offset）**：指向页框中的具体地址。

> 例：假设虚拟地址被分解为：
>
> - 10 位：页目录索引
> - 10 位：页表索引
> - 12 位：页内偏移

System Registers 系统寄存器

系统寄存器是 CPU 的一部分，用于控制处理器的各种系统级功能，包括**内存管理、特权级控制、中断处理和调试支持**。不同类型的寄存器负责不同的系统功能，主要包括：

1. **EFLAGS 寄存器**：控制任务和模式切换、中断处理以及访问权限。
2. **控制寄存器（CR0、CR2、CR3、CR4）**：管理内存分页和特定功能支持。
3. **调试寄存器（Debug Registers）**：用于设置断点和调试程序。
4. **描述符表寄存器（GDTR、LDTR、IDTR、TR）**：用于内存管理和任务切换。
5. **专用寄存器（Model-Specific Registers，MSRs）**：控制特定的 CPU 功能，如性能监控和内存类型范围。
    - MSRs 是与具体处理器型号相关的寄存器，主要用于操作系统和执行级代码（即**特权级 0**）的访问。
    - MSRs 控制以下内容：

- 调试扩展：如调试模式支持。
- 性能监控计数器：用于测量 CPU 的性能指标。
- 机器检查体系结构（MCA）：检测和报告硬件错误。
- 内存类型范围寄存器（MTRRs）：控制不同地址范围的内存类型（如缓存策略）。

1.2 实模式和保护模式转换

实模式

概述

- 实模式模拟了 Intel 8086 处理器的编程环境，并且处理器启动时默认进入此模式。
- 它可以访问的内存地址空间为 1MB，超过 1MB 的部分称为扩展内存，需要切换到保护模式后才能访问。

特性

- 地址访问方式：实模式中的地址由段基址：段内偏移地址组成，物理地址由段寄存器的值（乘以 16，左移 4 位）加上偏移量计算得出（20 位地址）。
  - 最大寻址空间：1MB（2^20 字节）。
  - 最大段长度：64KB。
- 缺乏内存保护：所有程序可以访问所有内存地址，容易出现冲突或数据破坏。
- 典型用途：
  - 主要用于系统初始化（如 BIOS 阶段）。
  - 在运行旧版 DOS 程序时使用。

保护模式

概述

- 保护模式是 x86 处理器的主要操作模式，支持内存保护、多任务和虚拟内存管理。
- 处理器大部分时间（99.99%）运行在此模式中。

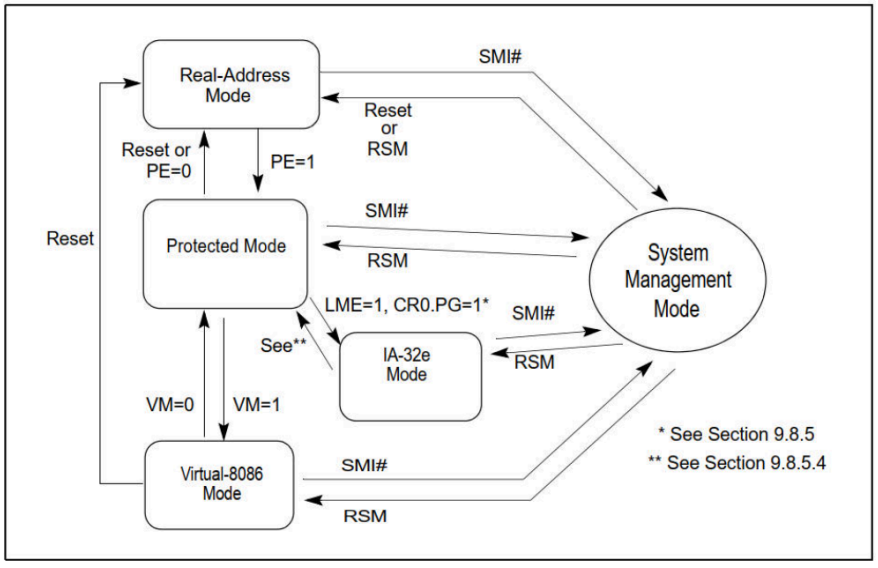
特性

- 内存管理：
  - 支持 段模式 和 分页模式。

    - 段模式是必需的，分页机制可以选择开启。分页机制将线性地址映射到物理地址，实现虚拟内存。
- 多任务支持：通过 TSS（任务状态段） 实现任务切换。
- 内存保护：使用描述符表（如 GDT 和 LDT）分配和管理各程序的地址空间，防止程序间互相干扰。
- 特权级管理：处理器有 4 个特权级（0-3），用于控制不同代码的访问权限。
- 最大内存访问空间：支持 32 位地址，最大可访问 4GB 内存。

转换步骤

> 来自于 https://1nvisble.github.io/2022/04/01/%E7%B3%BB%E7%BB%9F%E7%BA%A7%E4



> *从实模式切换到保护模式*
>
> 1. 准备数据结构和寄存器：
>   - 加载并初始化GDT、IDT、TSS、LDT 等数据结构。
>   - 若启用分页机制，需要设置页目录和页表。
> 2. 初始化系统寄存器：
>   - 使用 LGDT 指令加载 GDT 的基地址到 GDTR。
>   - 如果使用 LDT，执行 LLDT 指令加载 LDT 的段选择器到 LDTR。
>   - 执行 LTR 指令将任务选择子加载到 TR（任务寄存器）。
> 3. 启用保护模式：
>   - 禁用中断（使用 CLI 指令）。
>   - 设置 CR0 寄存器中的 PE 位（Protection Enable） 为 1，启用保护模式。

- 执行 FAR JMP 或 FAR CALL 指令，使指令流水线刷新，并加载新的代码段。

4. 启用中断：
   - 初始化并加载 IDT。
   - 使用 STI 指令重新启用可屏蔽硬件中断。

从保护模式切换回实模式

1. 关闭分页机制（如果启用过）：
   - 将程序控制转移到物理地址和线性地址相同的内存段。
   - 重置 CR0 寄存器的 PG 位（Paging Enable）。
2. 重置系统寄存器：
   - 清空 CR3 寄存器（页表目录地址）。
   - 使用 LIDT 指令加载实模式下的 IDT。
3. 切换回实模式：
   - 将 CR0 寄存器的 PE 位重置为 0。
   - 执行 FAR JMP 指令，刷新流水线，进入实模式代码段。
4. 启用中断：
   - 使用 STI 指令重新启用可屏蔽硬件中断。

1.3. 80x86 系统指令寄存器

标志寄存器 EFLAGS

下图为在标志寄存器中的系统标志，上部分为 EFLAGS Register

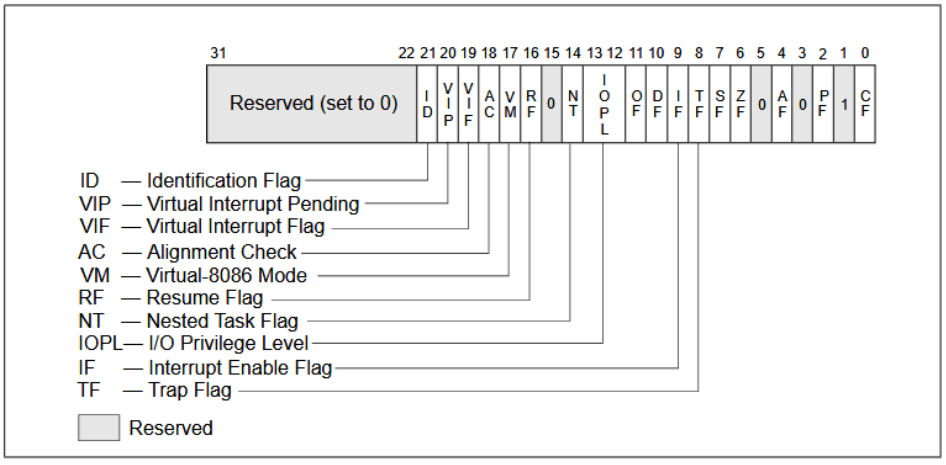


Figure 2-5. System Flags in the EFLAGS Register

EFLAGS 寄存器 是 x86 处理器的重要寄存器之一，保存了当前系统和程序的状态信息。它的系统标志和 I/O 权限级字段（IOPL）控制着中断处理、调试、任务切

换、虚拟 8086 模式以及内存对齐检查等系统操作。

以下是其中重要的系统标志：

1. TF（Trap Flag）——单步调试标志
2. IF（Interrupt Flag）——中断使能标志
3. IOPL（I/O Privilege Level）——I/O 权限级别
4. NT（Nested Task）——嵌套任务标志
5. RF（Resume Flag）——恢复标志
6. VM（Virtual-8086 Mode）——虚拟 8086 模式标志
7. AC（Alignment Check）——内存对齐检查标志
8. VIF（Virtual Interrupt Flag）——虚拟中断标志
9. VIP（Virtual Interrupt Pending）——虚拟中断挂起标志
10. ID（Identification Flag）——识别标志

介绍两个较为关键的标志

1. IF（Interrupt Flag） - 中断使能标志

- 作用：控制处理器对**可屏蔽硬件中断**的响应。
  - IF = 1：允许处理器响应硬件中断。
  - IF = 0：屏蔽可屏蔽中断，但**非屏蔽中断（NMI）**和**异常**仍会被处理。
- **修改限制**：CLI（关闭中断）、STI（开启中断）指令或 POPF、IRET 指令可以修改 IF 标志，但需要满足**特权级（CPL ≤ IOPL）**的要求。
- **使用场景**：操作系统在执行关键代码段时会禁用中断，以避免中断打断执行。

2. TF（Trap Flag） - 单步调试标志

- 作用：启用**单步调试模式**。
  - TF = 1：处理器在每条指令执行后都会触发一次**调试异常（#DB）**，供调试器检查程序状态。
  - TF = 0：禁用单步调试。
- **使用场景**：在调试模式下，通过逐条指令执行，开发者可以查看寄存器和内存状态，帮助定位错误。

内存管理寄存器

x86 处理器中提供了**四个内存管理寄存器**，用于管理段式内存结构。这些寄存器分别是：GDTR（**全局描述符表寄存器**）、LDTR（**本地描述符表寄存器**）、IDTR（中断

描述符表寄存器） 和 TR（任务寄存器）。它们指向关键的数据结构，这些结构控制着分段内存管理。

全局描述符表寄存器 GDTR（Global Descriptor Table Register (GDTR)）

GDTR 寄存器用于存放全局描述符表的线性基地址（*保护模式下 32 位，IA-32e 模式下 64 位*）和16 位的 GDT 表限长值。

- 基地址：指定 GDT 表中字节 0 在线性地址空间中的地址。
- 表限制长度：指明 GDT 表的字节长度值。
- LGDT、SGDT指令分别用于加载和保存 GDTR 中的值。
- 在机器刚通电或处理器复位后，基地址被默认设置为0，而长度被设置为0xFFFF。
- 在保护模式初始化过程中，必须为 GDTR 加载一个新的值。

局部描述符表寄存器 LDTR（Local Descriptor Table Register (LDTR)）

LDTR 寄存器用于存放局部描述符表的 16 位段选择子、基地址（*保护模式下 32 位，IA-32e 模式下 64 位*）、段限长和描述符属性值。

- 基地址：指明 LDT 段中字节 0 在线性地址空间中的地址。
- 段限长度：指明 LDT 段的字节数目。
- LLDT、SLDT指令分别用于加载和保存 LDTR 寄存器中段描述符的部分。
- 包含 LDT 表的段必须在 GDT 表中有一个段描述符项。
- 当使用 LLDT 指令把含有 LDT 表段的选择符加载进 LDTR 时，LDT 段描述符的段基址、段限长度以及描述符属性会被自动加载进 LDTR。
- 任务切换时，新任务的LDT 段的段选择子和段描述符会自动加载到 LDTR中。在写入新的 LDT 信息之前，LDTR 的内容不会被自动保存。
- 当机器通电或处理器复位后，段选择符和基地址被默认设置为0，而段长度被设置成0xFFFF。

中断描述符表寄存器 IDTR（Interrupt Descriptor Table Register (IDTR)）

IDTR 寄存器用于存放中断描述符表的线性基地址（*保护模式下 32 位，IA-32e 模式下 64 位*）和16 位的中断描述符表限长值。

- 基地址：指定 IDT 表中字节 0 在线性地址空间中的地址。
- 表限制长度：指明 IDT 表的字节数目。
- LIDT、SIDT指令分别用于加载和保存 IDTR 中的值。
- 在机器刚通电或处理器复位后，基地址被默认设置为0，而长度被设置为0xFFFF。
- IDTR 寄存器中的基址和限长可以在处理器初始化过程中进行更改。

任务寄存器 TR（Task Register（TR））

TR 寄存器存放了当前任务 TSS 段的 16 位段选择子、基地址（*保护模式下 32 位，IA-32e 模式下 64 位*）、段长度和描述符属性值。

- TR 引用 GDT 表中的一个 TSS 类型的描述符。
- 基地址：指定 TSS 段中字节 0 在线性地址空间中的地址。
- 段长度：指明 TSS 的字节数目。
- LTR、STR指令分别用于加载和保存 TR 寄存器的段选择符部分。
- 当LTR 指令把选择符加载进任务寄存器时，TSS 描述符中的段基址、段限长以及描述符属性会被自动加载到 TR中。
- 当机器通电或处理器复位后，段选择符和基地址被默认设置为0，而段长度被设置成0xFFFF。
- 在任务切换时，处理器会把新任务的 TSS 的段选择子和段描述符自动加载入 TR中。在写入新的 TSS 信息之前，TR 的内容不会被自动保存

控制寄存器

完整的控制寄存器

patibility mode.

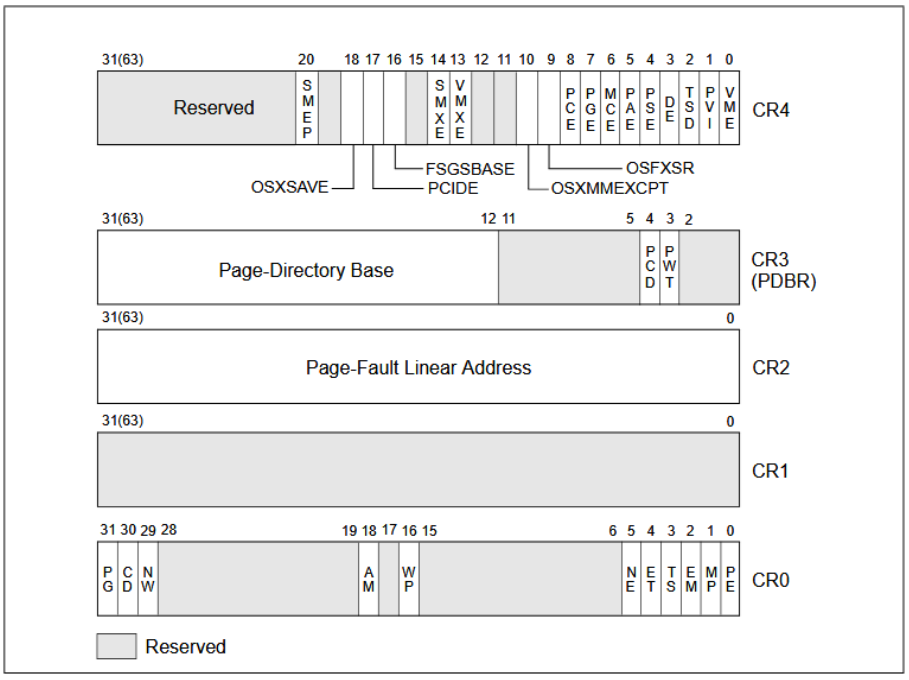


Figure 2-7. Control Registers

1. CR0—系统控制标志寄存器

- 作用：控制处理器的操作模式和状态。
- 关键位：
  - PE（Protection Enable，位 0）：启用保护模式（PE=1）。
  - PG（Paging Enable，位 31）：启用分页机制（PG=1）。
  - ET（Extension Type，位 4）：指示协处理器类型。
  - WP（Write Protect，位 16）：控制内核模式下是否能写入只读页面。
  - NE（Numeric Error，位 5）：启用 x87 浮点异常。
- 初始化：
  在保护模式初始化过程中，必须设置PE 位以进入保护模式，并可选择设置PG 位启用分页。

2. CR1 — 保留寄存器

- 作用：该寄存器目前保留，不用于任何功能。写入此寄存器会导致异常。

3. CR2 — 缺页异常线性地址寄存器

- 作用：存储引发缺页异常（Page Fault）的线性地址。
  - 当处理器发生缺页异常时，CR2 会记录导致异常的地址，方便操作系统的异常处理程序进行恢复。

4. CR3 — 页表基址寄存器

- 作用：存储分页结构的基址，并包含两个控制标志（PCD 和 PWT）。
  - 基地址：包含页表目录或其他分页层次结构的*物理基地址*。地址的低 12 位固定为 0，因此页表必须对齐到4KB 页边界。
  - PCD（Page Cache Disable，位 4）：禁用该分页结构的缓存。
  - PWT（Page Write-Through，位 3）：控制该分页结构是否启用直写缓存（write-through）。
- 分页模式：
  - 32 位模式：CR3 包含页目录的基址。

SUMMARY

寄存器	功能	关键用途
CR0	系统控制标志寄存器	启用保护模式和分页，控制系统状态
CR1	保留寄存器	暂无功能，写入将触发异常
CR2	缺页异常地址	存储导致缺页异常的线性地址
CR3	页表基址寄存器	存储页表目录的物理基址，控制缓存行为

1.4 系统指令

系统指令主要负责处理系统级函数，如加载系统寄存器、管理缓存、管理中断或设置调试寄存器。其中许多指令只能由操作系统软件或者处于特权级别 0 的程序执行。其他指令可以在任何特权级别执行，因此应用程序也可使用。

给出常用系统指令的总结：

指令	全名	受保护	描述
LGDT	Load GDTR Register	是	从内存加载 GDT 的基址和限长到 GDTR
SGDT	Store GDTR Register	否	将 GDTR 中的内容存储到内存
LIDT	Load IDTR Register	是	从内存加载 IDT 的基址和限长到 IDTR
SIDT	Store IDTR Register	否	将 IDTR 中的内容存储到内存
LLDT	Load LDT Register	是	从内存加载 LDT 的段选择子和描述符到 LDTR

指令	全名	受保护	描述
SLDT	Store LDT Register	否	将 LDTR 中的 LDT 段选择子存储到内存或寄存器
LTR	Load Task Register	是	加载 TSS 的段选择子和描述符到 TR
STR	Store Task Register	否	将 TR 中的 TSS 段选择子和描述符存储到内存或寄存器