The Setup:
Paving the path to the OS

Prof. Dr. Antônio Augusto Fröhlich

UFSC / LISHA

https://lisha.ufsc.br/~guto

Almost there at the OS

- BIOS brought the system on
  - BIST, POST, hooks
  - First instruction fetched – 0x7c00
  - Lots of “jmp” so far, no calls, no stack
- Bootstrap loaded the OS
  - 16-bit assembly code
  - rudimentary process model
  - Configured memory and bus (A20)
  - Entered 32-bit Protected Mode (for GCC code)
- Setup
  - Bring the machine into a usable state

The Role of SETUP

- It “sets up” extremely architecture-dependent data structures
  - For the x86 PC
    - IDT – Interrupt Descriptor Table
    - GDB – Global Descriptor Table
    - Paging
    - PCI
    - FPU
    - Timers and interrupts
    - etc...
- Delivers
  - A basic memory model (flat)
  - A basic process model (single task)

System Information
Boot Map

```
struct Boot_Map
{
    Phy_Addr mem_base;          // Memory base
    unsigned int mem_size;      // Memory size (in bytes)
    int cpu_type;               // Processor type
    int cpu_clock;              // Processor clock frequency in Hz
    int n_threads;              // Max number of threads
    int n_tasks;                // Max number of tasks
    unsigned short host_id;     // The local host id (-1 => RARP)
    unsigned short n_nodes;     // Number of nodes in SAN
    int img_size;               // Boot image size in bytes
    int setup_off;              // SETUP offset in the boot image
    int system_off;             // OS offset in the boot image
    int loader_off;             // LOADER offset in the boot image
    int app_off;                // APPs offset in the boot image
};
```

System Information
Physical Memory Map

```
struct Physical_Memory_Map
{
  Phy_Addr app_lo;          // Application memory's lowest address
  Phy_Addr app_hi;          // Application memory's highest address
  Phy_Addr int_vec;         // Interrupt Vector
  Phy_Addr sys_pt;          // System Page Table
  Phy_Addr sys_pd;          // System Page Directory
  Phy_Addr sys_info;        // System Info
  Phy_Addr phy_mem_pts;     // Page tables to map the whole phy memory
  Phy_Addr io_mem_pts;      // Page tables to map the IO address space
  Phy_Addr sys_code;        // OS Code Segment
  Phy_Addr sys_data;        // OS Data Segment
  Phy_Addr sys_stack;       // OS Stack Segment
  Phy_Addr free;            // Free memory base
  unsigned int free_size;   // Free memory size (in frames)
  Phy_Addr mach1;           // Machine specific entries
  Phy_Addr mach2;
  Phy_Addr mach3;
  Phy_Addr free1_base;      // First free memory chunk base address
  Phy_Addr free1_top;       // First free memory chunk top address
  Phy_Addr free2_base;      // Second free memory chunk base address
  Phy_Addr free2_top;       // Second free memory chunk top address
};
```

System Information
Logical Memory Map

```
struct Logical_Memory_Map
{
  Log_Addr base;          // Lowest valid logical address
  Log_Addr top;           // Highest valid logical address
  Log_Addr app_lo;        // Application memory lowest address
  Log_Addr app_entry;     // First application's entry point
  Log_Addr app_code;      // First application's code base address
  Log_Addr app_data;      // First application's data base address
  Log_Addr app_hi;        // Application memory highest address
  Log_Addr phy_mem;       // Whole physical memory (contiguous)
  Log_Addr io_mem;        // IO address space
  Log_Addr int_vec;       // Interrupt Vector
  Log_Addr sys_pt;        // System Page Table
  Log_Addr sys_pd;        // System Page Directory
  Log_Addr sys_info;      // System Info
  Log_Addr sys_code;      // OS Code Segment
  Log_Addr sys_data;      // OS Data Segment
  Log_Addr sys_stack;     // OS Stack Segment
  Log_Addr mach1;         // Machine specific entries
  Log_Addr mach2;
  Log_Addr mach3;
};
```

System Information
I/O Memory Map

```
struct IO_Memory_Map
{
  int locator;
  Phy_Addr phy_addr;
  Log_Addr log_addr;
  unsigned int size;
};
```

System Information

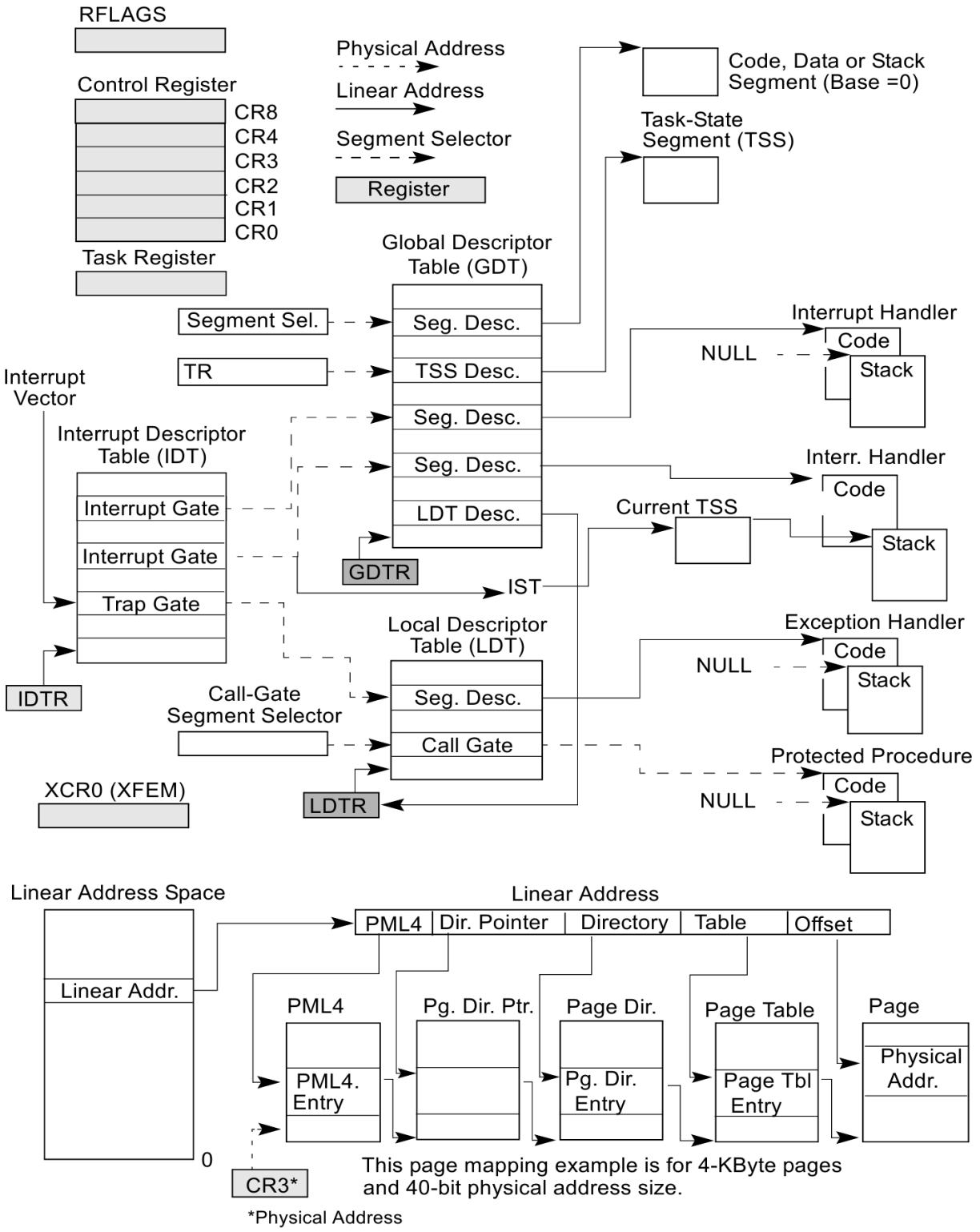
```
class System_Info
{
public:
    typedef unsigned int Log_Addr;
    typedef unsigned int Phy_Addr;

    unsigned int mem_size;   // Memory size (in pages)
    unsigned int mem_free;   // Free memory (in pages)
    unsigned int iomm_size;
    Boot_Map bm;
    Physical_Memory_Map pmm;
    Logical_Memory_Map lmm;
    IO_Memory_Map iomm[];
};
```

Starting SETUP

- SETUP begins running where the bootstrap loaded it
  - Outside the address space it was compiled for
  - Using only PC-relative addressing
  - But it has access to `System_Info`
  - Boot image at `BOOT_IMAGE_ADDR`
- Verify system image
  - Validate ELF image
  - Check if ELF entry point is accessible
- Relocate the SETUP (as it runs)
  - Move boot image to after SETUP
  - Setup a single-page stack for SETUP
- Call actual SETUP main

System Memory

- Allocate (reserve) memory for all entities we have to setup
  - Start at the highest address
  - Flat memory model
  - Reserve memory for
    - IDT – Interrupt Descriptor Table
    - GDT – Global Descriptor Table
    - SysPT
    - SysPD
    - System_Info
    - IO space
    - OS code, data, and stack
    - Mark rest of memory as free to applications


RFLAGS
Control Register
CR8
CR4
CR3
CR2
CR1
CR0
Task Register
Physical Address
Linear Address
Segment Selector
Register
Code, Data or Stack Segment (Base =0)
Task-State Segment (TSS)
Global Descriptor Table (GDT)
Segment Sel.
TR
Seg. Desc.
TSS Desc.
Seg. Desc.
Seg. Desc.
LDT Desc.
GDTR
Interrupt Handler
Code
Stack
NULL
Interrupt Vector
Interrupt Descriptor Table (IDT)
Interrupt Gate
Interrupt Gate
Trap Gate
IDTR
IST
Interr. Handler
Code
Stack
Current TSS
Local Descriptor Table (LDT)
Seg. Desc.
Call Gate
LDTR
Call-Gate Segment Selector
Exception Handler
Code
Stack
NULL
Protected Procedure
Code
Stack
NULL
XCR0 (XFEM)
Linear Address Space
Linear Addr.
0
Linear Address
PML4
Dir. Pointer
Directory
Table
Offset
PML4
PML4. Entry
Pg. Dir. Ptr.
Page Dir.
Pg. Dir. Entry
Page Table
Page Tbl Entry
Page
Physical Addr.
CR3*
This page mapping example is for 4-KByte pages and 40-bit physical address size.
*Physical Address

x86 GDT

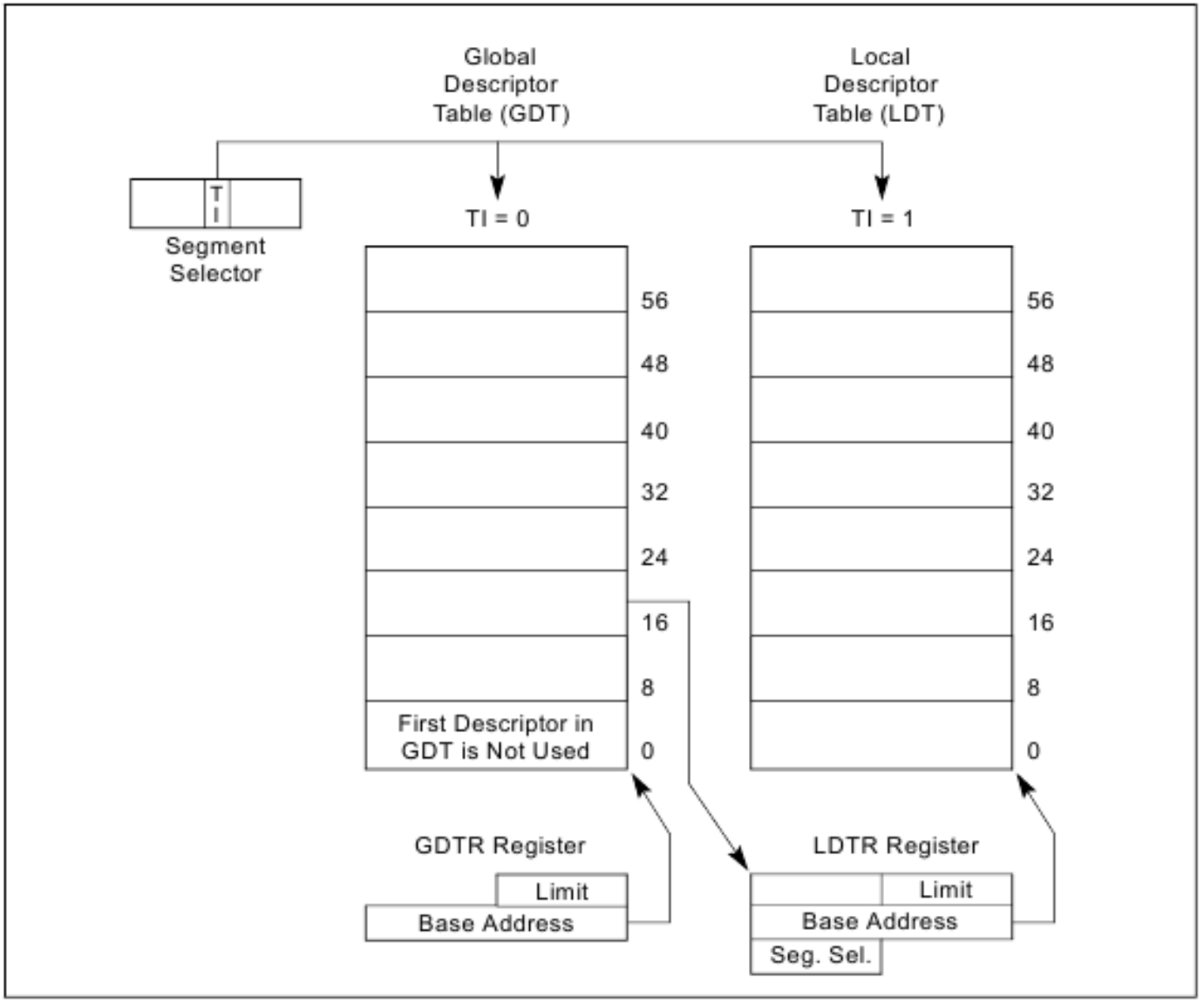
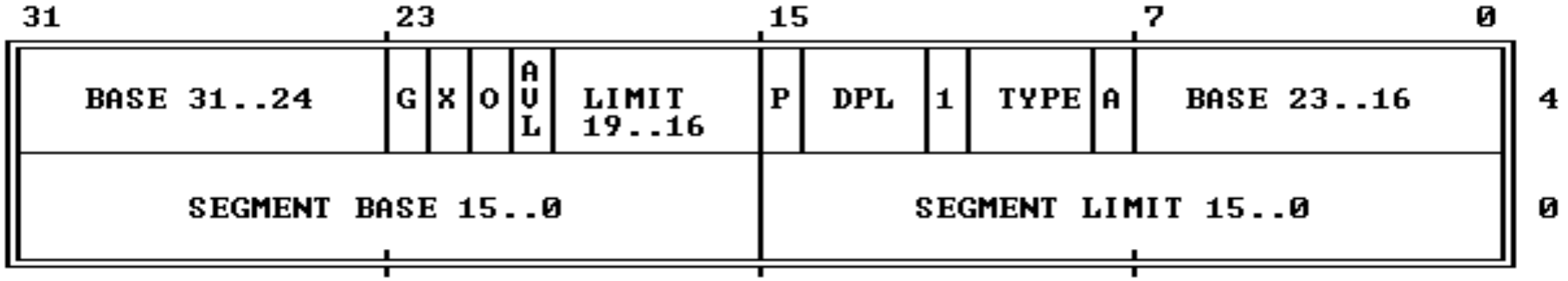


Figure 3-10. Global and Local Descriptor Tables

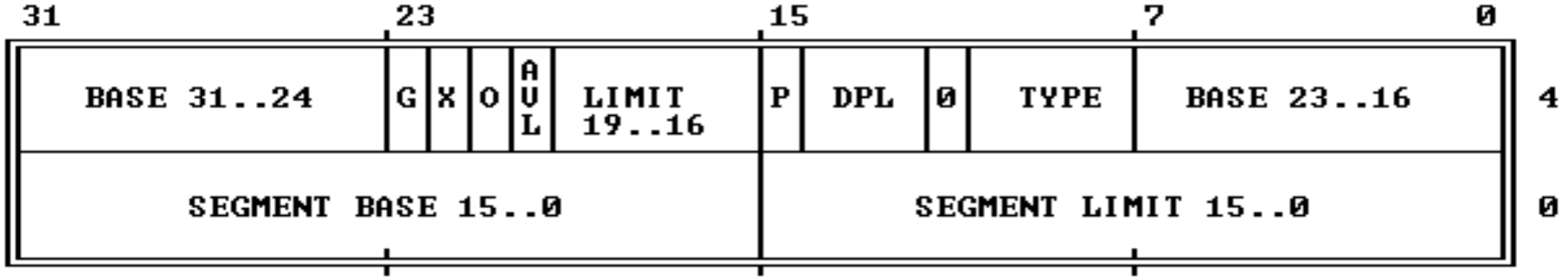
x86 GDT entries

Figure 5-3. General Segment-Descriptor Format

DESCRIPTORS USED FOR APPLICATIONS CODE AND DATA SEGMENTS

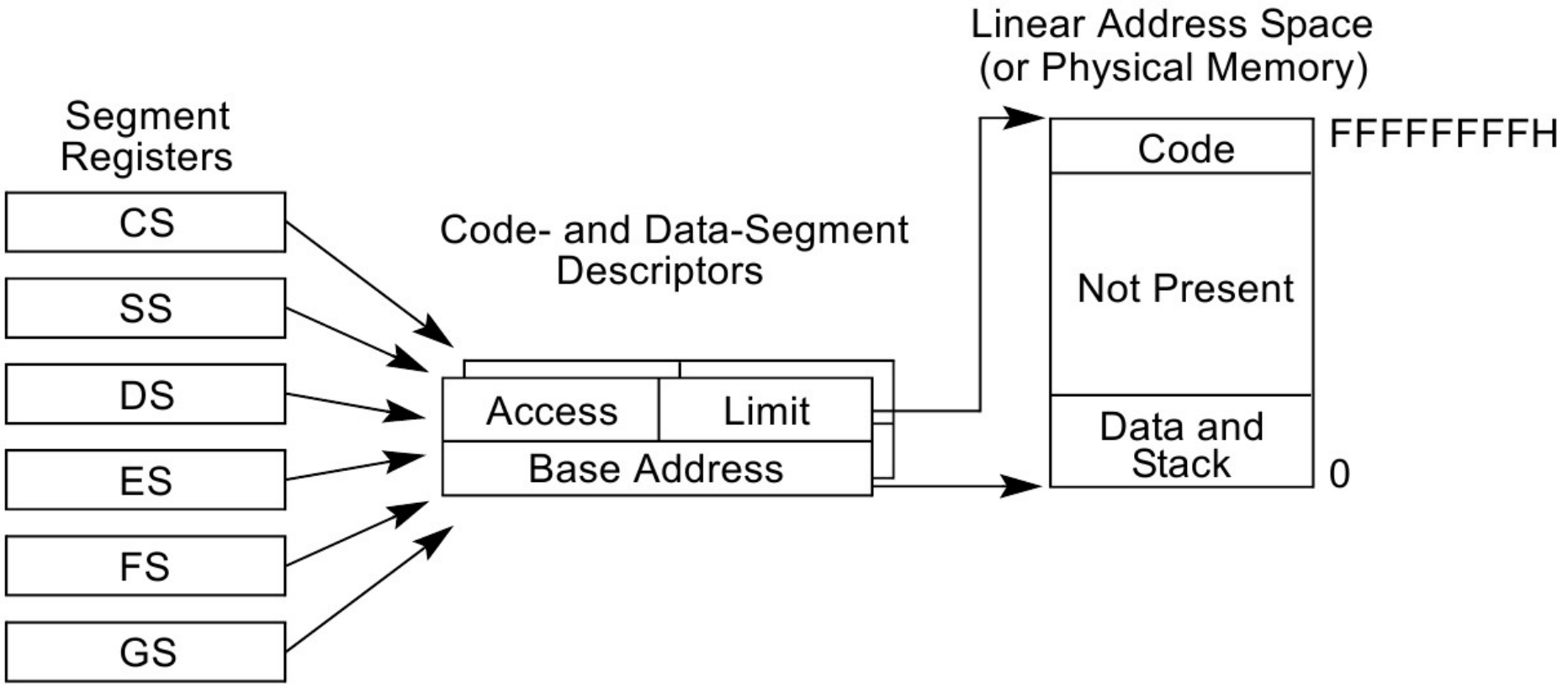


DESCRIPTORS USED FOR SPECIAL SYSTEM SEGMENTS



A – ACCESSED
AVL – AVAILABLE FOR USE BY SYSTEMS PROGRAMMERS
DPL – DESCRIPTOR PRIVILEGE LEVEL
G – GRANULARITY
P – SEGMENT PRESENT

x86 Flat Memory Model



x86 Paging

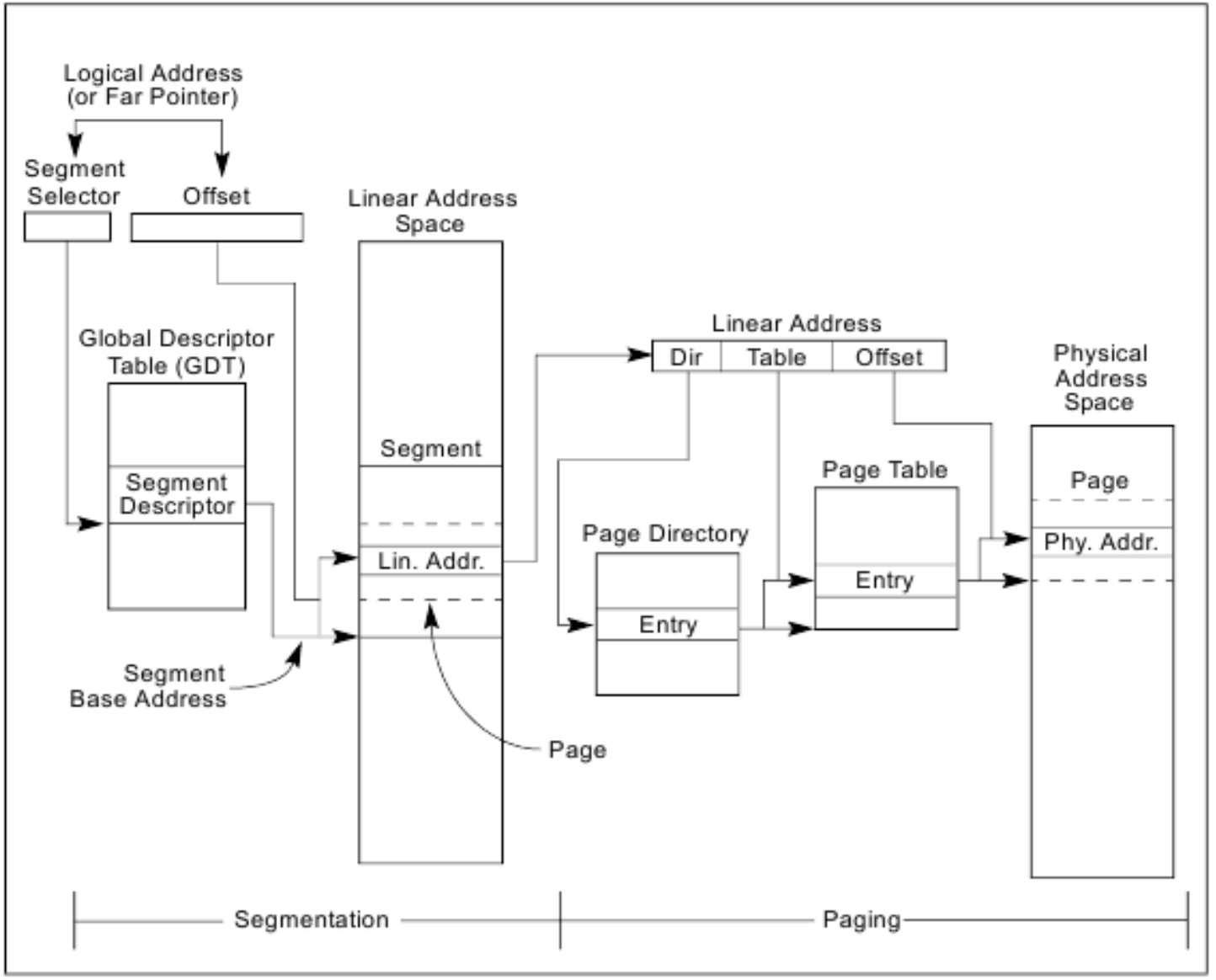
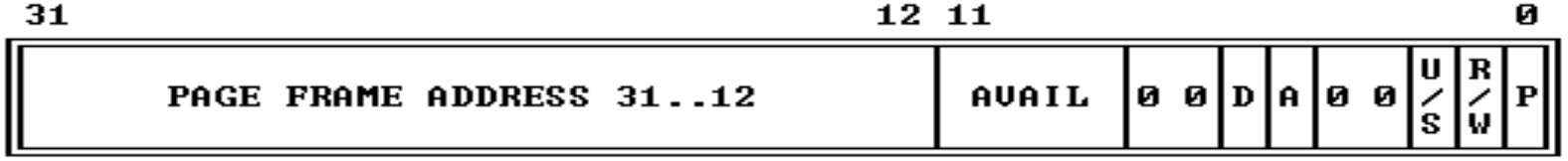


Figure 3-1. Segmentation and Paging

x86 Page Tables

Figure 5-8. Format of a Linear Address

Figure 5-10. Format of a Page Table Entry



P — PRESENT
R/W — READ/WRITE
U/S — USER/SUPERVISOR
D — DIRTY
AVAIL — AVAILABLE FOR SYSTEMS PROGRAMMER USE

NOTE: 0 INDICATES INTEL RESERVED. DO NOT DEFINE.

x86 Pate Tables by SETUP

- System Page Directory (SysPD)
  - First level page table
    - Point to 2nd level page tables
- Page Tables
  - Map system code, data, and stack
  - Map all the Physical Memory
- Mapping performed by MMU

PCI Bus

- Scan the PCI bus
  - Look for devices with memory mapped regions
- Fill the IO_Memory_Map in System_Info

```
for(unsigned int i = 0; i < si->iomm_size; i++)
  si->iomm[i].log_addr = MM::IO_MEM + (si->iomm[i].phy_addr - base);
```

APIC

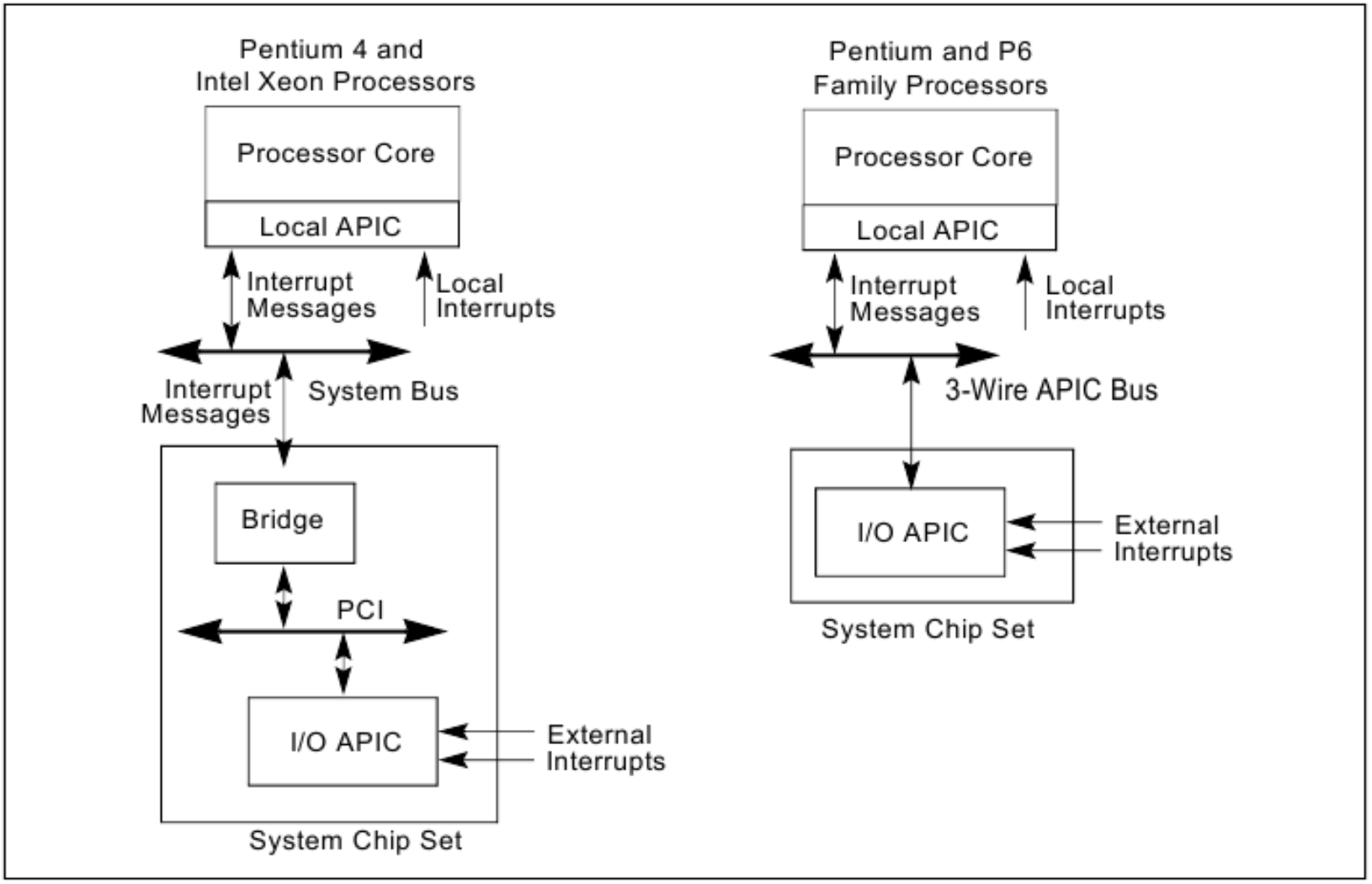


Figure 10-1. Relationship of Local APIC and I/O APIC In Single-Processor Systems

Intel 3A

Local APIC

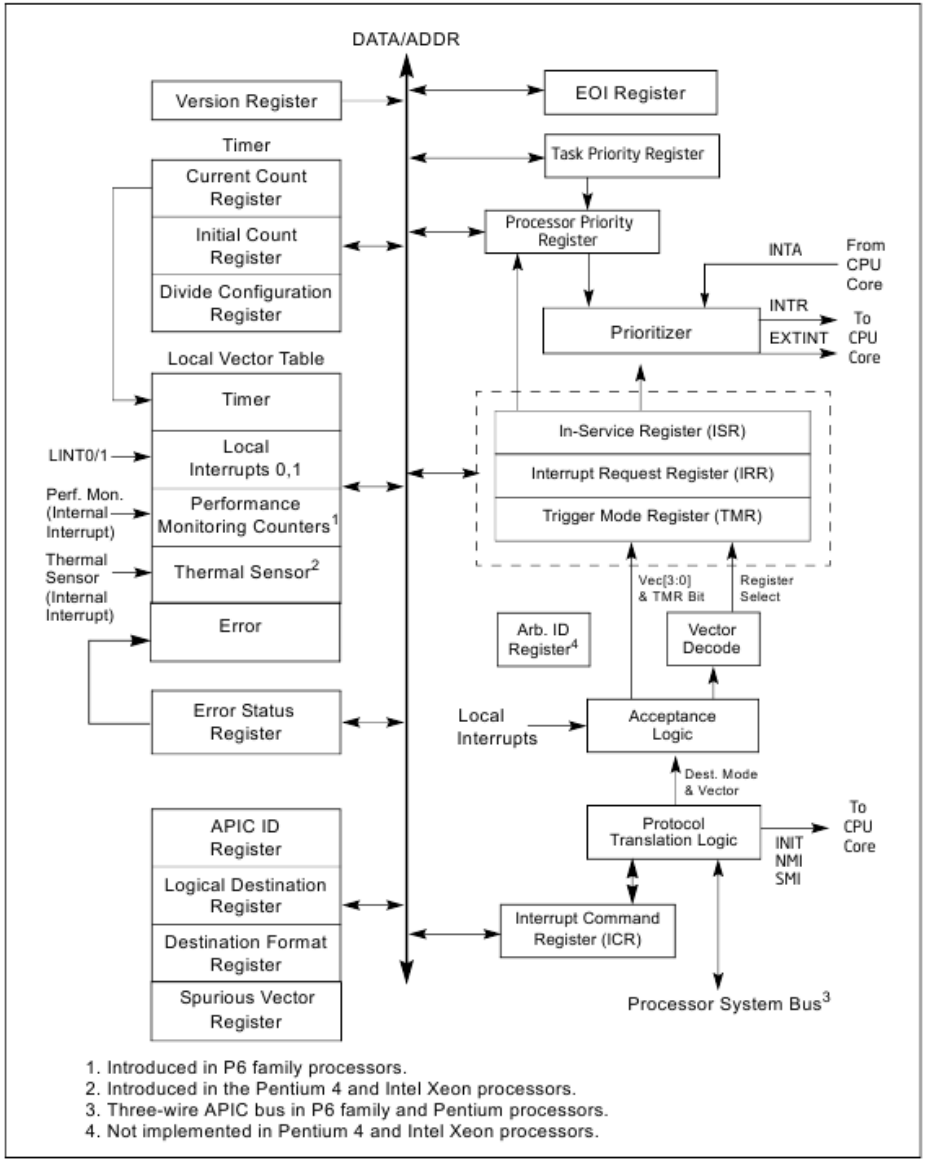


Figure 10-4. Local APIC Structure

IDT

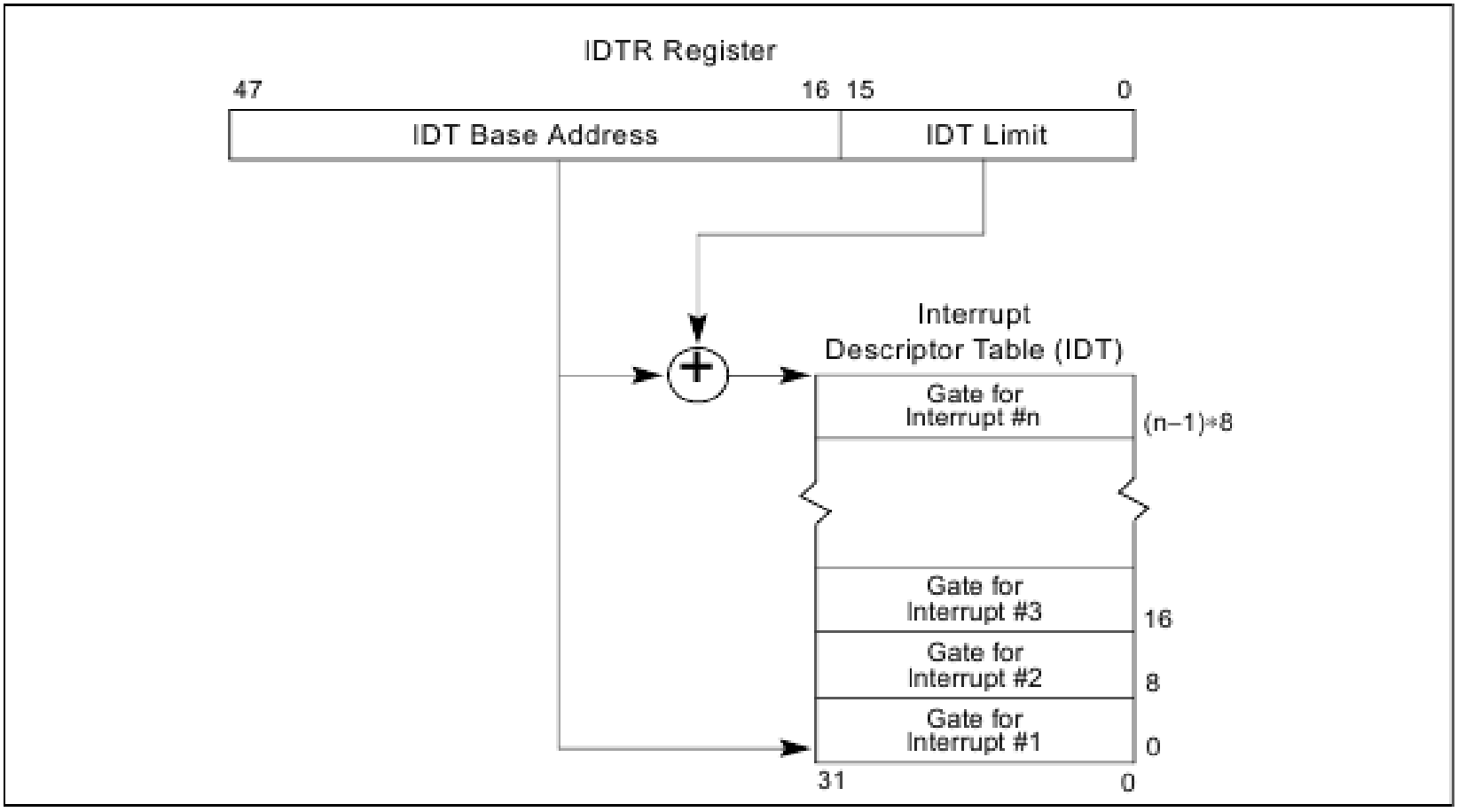


Figure 6-1. Relationship of the IDTR and IDT

Finishing SETUP

- Make previous configuration active
  - Set IDTR
  - Set GDTR
  - Set CR3 (page directory)
  - Enable paging
  - Break pre-fetch queue to activate configuration
  - Reload segment registers (flat model)
  - Remap pointers to their logical addresses
    - Stack
    - System_Info
    - Flush TLB (MMU – make sure new config is in use)
- Load OS and application(s)
- Enable interrupts
- Call system/application