Operating System

Dr. GuoJun LIU

Harbin Institute of Technology

http://guojunos.hit.edu.cn

Basic

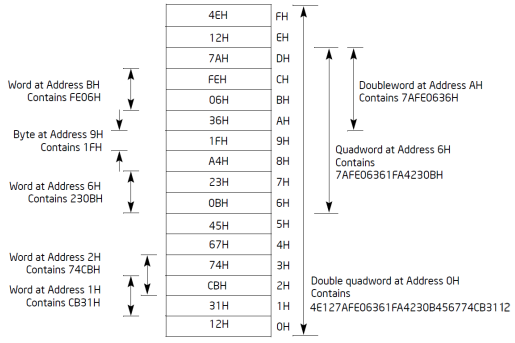
- Fundamental Data Types in Memory
- Registers
  - General System and Application Programming Registers
  - Special Uses of General-purpose Registers
  - Use of Segment Registers for Flat Memory Model
  - Use of Segment Registers in Segmented Memory Model
  - Default Segment Selection Rules
- EFLAGS Transfer Instructions
- Stack
  - Stack Structure
  - Setting Up a Stack
  - Calling Procedures Using Call And Ret
  - Stack Switch on a Call to a Different Privilege Level
  - Stack Usage on Transfers to Interrupt and Exception Handling Routines
- Conditional Jump Instructions
- I/O Permission Bit Map

Chapter A1

Intel IA-32 Architectures Software Developer's Manual

Intel IA-32 用户开发手册

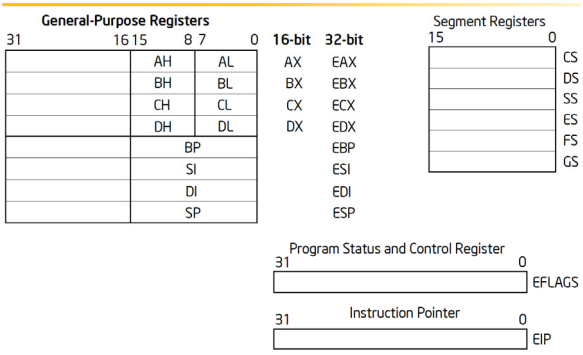
Fundamental Data Types in Memory



Outline

- Basic
- System Architecture Overview
- Protected-mode Memory Management
- Segment Descriptor
- Protection
- Interrupt And Exception Handling
- Task Management

General System and Application Programming Registers



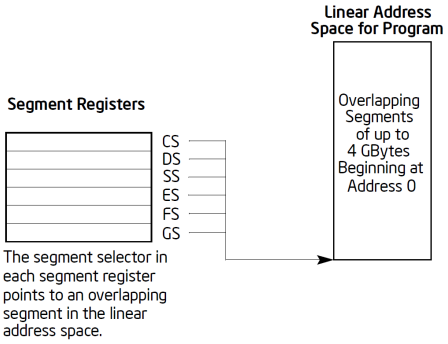
Special uses of general-purpose registers

- **EAX**
  - Accumulator for operands and results data
- **EBX**
  - Pointer to data in the DS segment
- **ECX**
  - Counter for string and loop operations
- **EDX**
  - I/O pointer
- **ESI**
  - Pointer to data in the segment pointed to by the DS register; source pointer for string operations
- **EDI**
  - Pointer to data (or destination) in the segment pointed to by the ES register; destination pointer for string operations
- **ESP**
  - Stack pointer (in the SS segment)
- **EBP**
  - Pointer to data on the stack (in the SS segment)

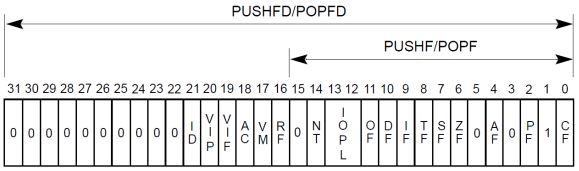
Default Segment Selection Rules

Reference Type	Register Used	Segment Used	Default Selection Rule
Instructions	CS	Code Segment	All instruction fetches.
Stack	SS	Stack Segment	All stack pushes and pops. Any memory reference which uses the ESP or EBP register as a base register.
Local Data	DS	Data Segment	All data references, except when relative to stack or string destination.
Destination Strings	ES	Data Segment pointed to with the ES register	Destination of string instructions.

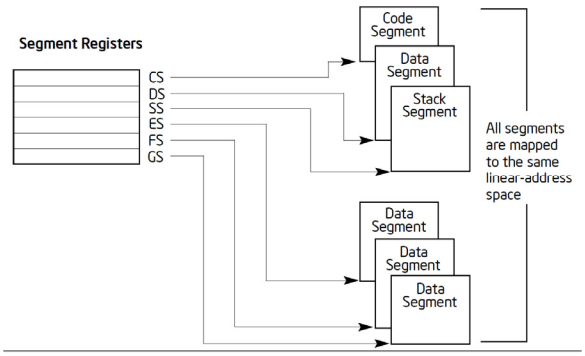
Use of Segment Registers for Flat Memory Model



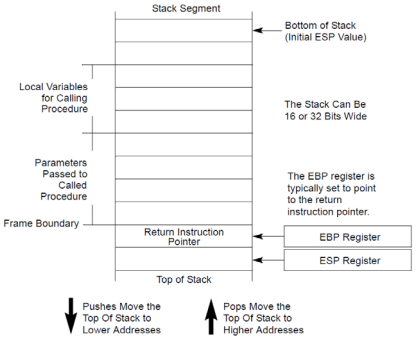
EFLAGS Transfer Instructions



Use of Segment Registers in Segmented Memory Model



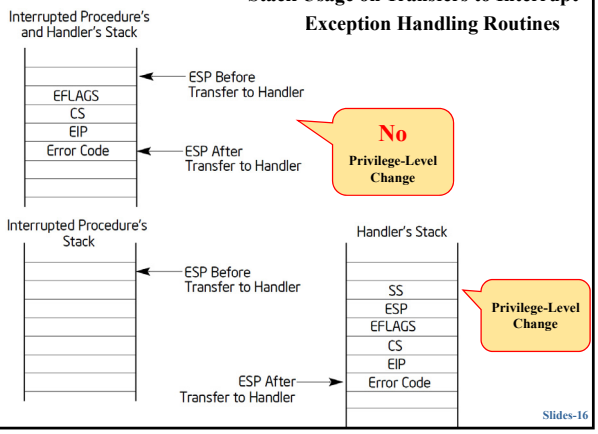
Stack Structure



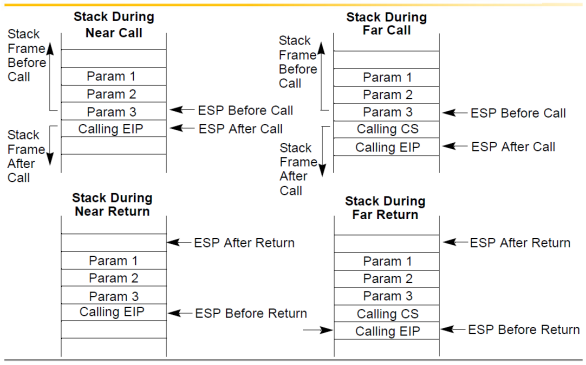
Setting Up a Stack

- **To set a stack and establish it as the current stack, the program or operating system/executive must do the following**
  - Establish a stack segment.
  - Load the **segment selector** for the stack segment into the **SS** register using a **MOV, POP, or LSS instruction**.
  - Load the **stack pointer** for the stack into the **ESP** register using a **MOV, POP, or LSS** instruction.
    - The LSS instruction can be used to load the SS and ESP registers in one operation

Stack Usage on Transfers to Interrupt Exception Handling Routines



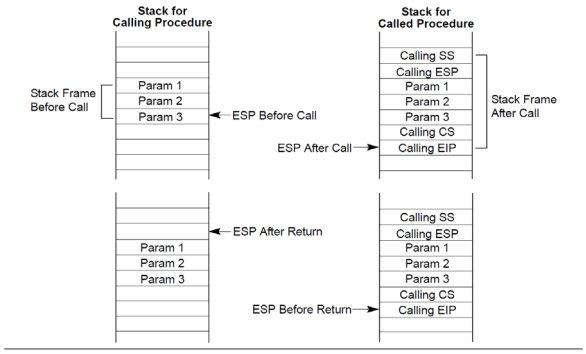
Calling Procedures Using Call And Ret



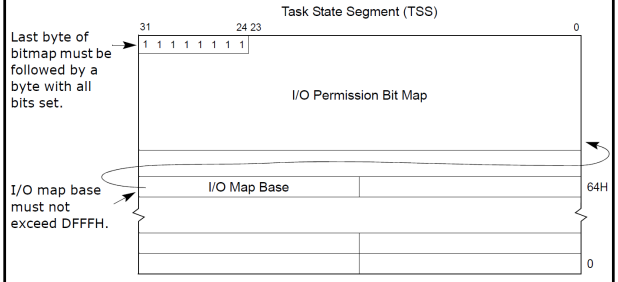
Conditional Jump Instructions

Instruction Mnemonic	Condition (Flag States)	Description
Unsigned Conditional Jumps		
JA/JNBE	(CF or ZF) = 0	Above/not below or equal
JAE/JNB	CF = 0	Above or equal/not below
JB/JNAE	CF = 1	Below/not above or equal
JBE/JNA	(CF or ZF) = 1	Below or equal/not above
JC	CF = 1	Carry
JE/JZ	ZF = 1	Equal/zero
JNC	CF = 0	Not carry
JNE/JNZ	ZF = 0	Not equal/not zero
JNP/JPO	PF = 0	Not parity/parity odd
JP/JPE	PF = 1	Parity/parity even
JCXZ	CX = 0	Register CX is zero
JECXZ	ECX = 0	Register ECX is zero
Signed Conditional Jumps		
JG/JNLE	((SF xor OF) or ZF) = 0	Greater/not less or equal
JGE/JNL	(SF xor OF) = 0	Greater or equal/not less
JL/JNGE	(SF xor OF) = 1	Less/not greater or equal
JLE/JNG	((SF xor OF) or ZF) = 1	Less or equal/not greater
JNO	OF = 0	Not overflow
JNS	SF = 0	Not sign (non-negative)
JO	OF = 1	Overflow
JS	SF = 1	Sign (negative)

Stack Switch on a Call to a Different Privilege Level



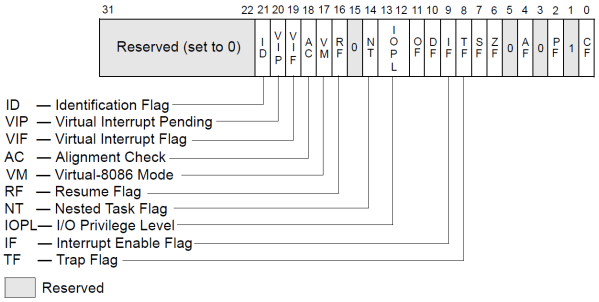
I/O Permission Bit Map

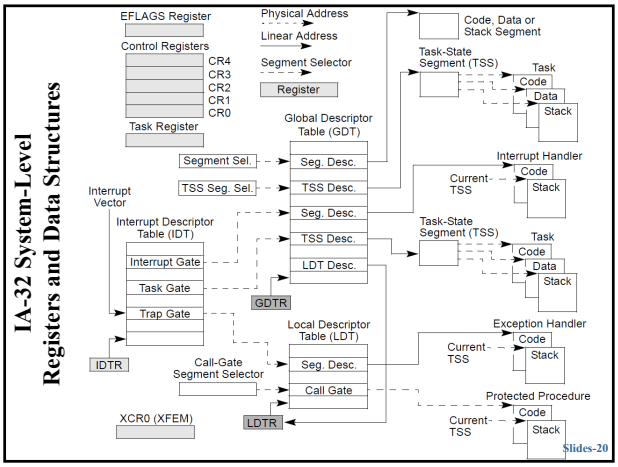


System Architecture Overview

- Overview of The System-level Architecture
- System Flags and Fields in the Eflags Register
- Memory-management Registers
- Control Registers
- System Instruction Summary

EFLAGS Register





Memory-management Registers

System Table Registers

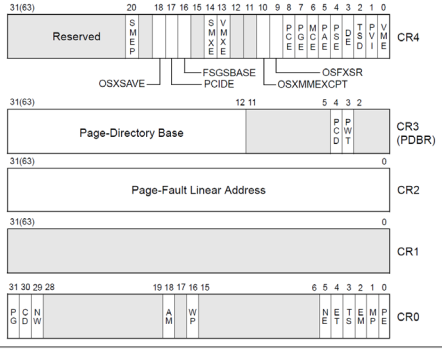
	47(79) — 16 15 — 0	
GDTR	32(64)-bit Linear Base Address	16-Bit Table Limit
IDTR	32(64)-bit Linear Base Address	16-Bit Table Limit

	System Segment Registers 15 — 0	Segment Descriptor Registers (Automatically Loaded)		
				Attributes
Task Register	Seg. Sel.	32(64)-bit Linear Base Address	Segment Limit	
LDTR	Seg. Sel.	32(64)-bit Linear Base Address	Segment Limit	

System Segments, Segment Descriptors, and Gates

- **Execution environment of a program or procedure**
  - code, data, and stack segments
  - the task-state segment (TSS)
  - the LDT
- **The GDT is not considered a segment**
  - it is not accessed by means of a segment selector and segment descriptor
- **IDT is not a segment**
- **A set of special descriptors called gates**
  - **call gates**
  - **interrupt gates**
  - **trap gates**
  - **task gates**
- **Provide protected gateways to system procedures and handlers**
  - that may operate at a different privilege level than application programs and most procedures.

Control Registers



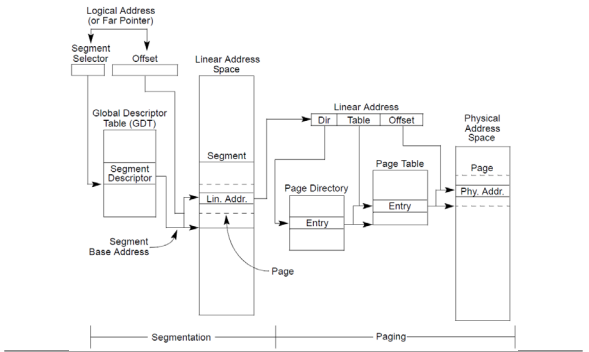
Instruction	Description	Useful to Application?	Protected from Application?
LLDT	Load LDT Register	No	Yes
SLDT	Store LDT Register	No	No
LGDT	Load GDT Register	No	Yes
SGDT	Store GDT Register	No	No
LTR	Load Task Register	No	Yes
STR	Store Task Register	No	No
LIDT	Load IDT Register	No	Yes
SIDT	Store IDT Register	No	No
MOV CRn	Load and store control registers	No	Yes
SMSW	Store MSW	Yes	No
LMSW	Load MSW	No	Yes
CLTS	Clear TS flag in CR0	No	Yes
MOV DRn	Load and store debug registers	No	Yes
HLT	Halt Processor	No	Yes

Summary of System Instructions

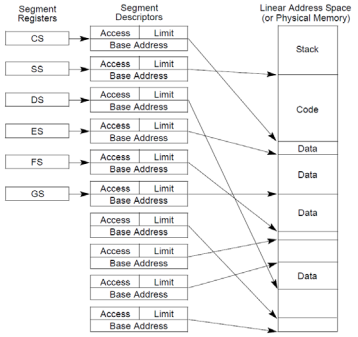
Protected-mode Memory Management

- Memory management overview
- Multi-Segment Model
- Logical Address to Linear Address Translation
- Linear Address to Physical Address Translation
- Segment Selector
- Segment Registers
- Global and Local Descriptor Tables (GDT and LDT)
- System Table Registers and System Segment Selector

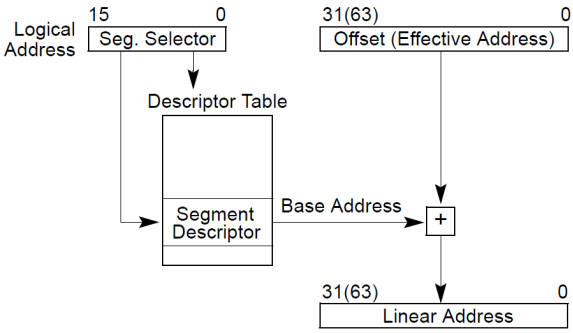
Memory Management Overview



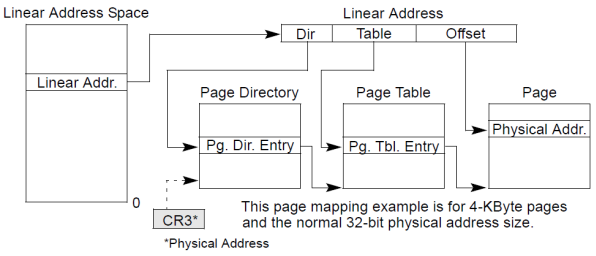
Multi-Segment Model



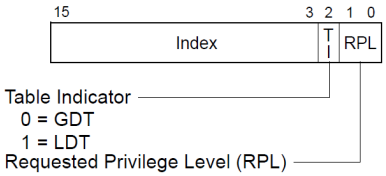
Logical Address to Linear Address Translation



Linear Address to Physical Address Translation



Segment Selector



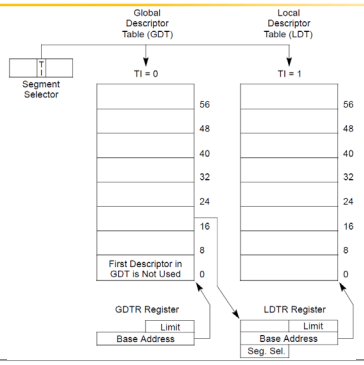
Segment Registers

Visible Part	Hidden Part	
Segment Selector	Base Address, Limit, Access Information	CS
		SS
		DS
		ES
		FS
		GS

Segment Registers

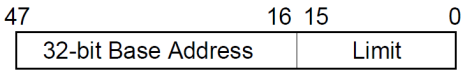
- **Two kinds of load instructions are provided for loading the segment registers**
  - **Direct load instructions** such as the **MOV**, **POP**, **LDS**, LES, **LSS**, LGS, and LFS instructions
    - These instructions explicitly reference the segment registers
  - **Implied load instructions** such as the far pointer versions of the **CALL**, **JMP**, and **RET** instructions, the SYSENTER and SYSEXIT instructions, and the **IRET**, **INTn**, **INTO** and **INT3** instructions.
    - These instructions change the contents of the CS register (and sometimes other segment registers) as an incidental part of their operation

Global and Local Descriptor Tables



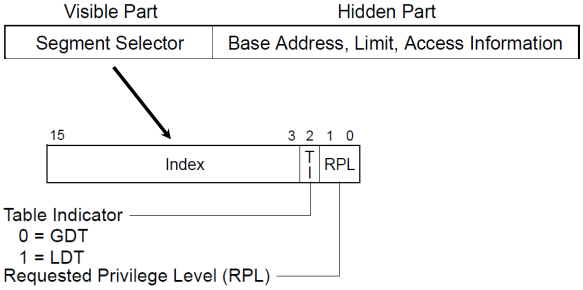
System Table Registers

GDTR **IDTR**



System Segment Selector

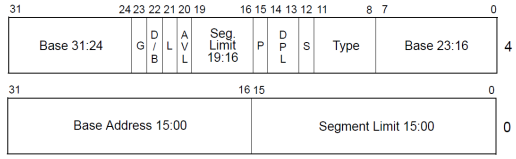
LDTR **TR**



Segment Descriptor

- Segment Descriptor
- Code- and Data-Segment Types
- System Descriptor Types
- Gate Descriptors
- Call-Gate Descriptor
- Call-Gate Mechanism

Segment Descriptor



L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

System Descriptor Types

- When the S (descriptor type) flag in a segment descriptor is clear, the descriptor type is a system descriptor
  - Local descriptor-table (LDT) segment descriptor
  - Task-state segment (TSS) descriptor
  - Call-gate descriptor
  - Interrupt-gate descriptor
  - Trap-gate descriptor
  - Task-gate descriptor

S=0

	Type Field				Description
Decimal	11	10	9	8	32-Bit Mode
0	0	0	0	0	Reserved
1	0	0	0	1	16-bit TSS (Available)
2	0	0	1	0	LDT
3	0	0	1	1	16-bit TSS (Busy)
4	0	1	0	0	16-bit Call Gate
5	0	1	0	1	Task Gate
6	0	1	1	0	16-bit Interrupt Gate
7	0	1	1	1	16-bit Trap Gate
8	1	0	0	0	Reserved
9	1	0	0	1	32-bit TSS (Available)
10	1	0	1	0	Reserved
11	1	0	1	1	32-bit TSS (Busy)
12	1	1	0	0	32-bit Call Gate
13	1	1	0	1	Reserved
14	1	1	1	0	32-bit Interrupt Gate
15	1	1	1	1	32-bit Trap Gate

S=0

System-Segment and Gate-Descriptor Types

	Type Field				Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

S=1

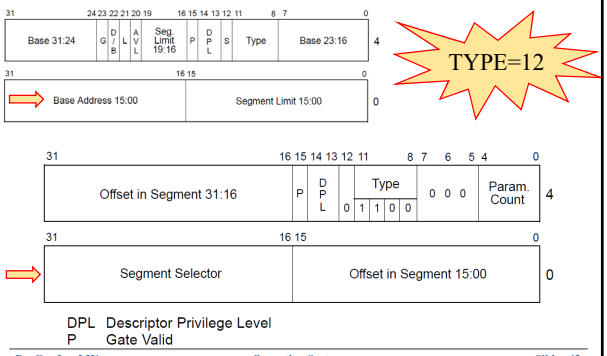
Code- and Data-Segment Types

Gate Descriptors

- To provide controlled access to code segments with different privilege levels, the processor provides special set of descriptors called gate descriptors
- There are four kinds of gate descriptors
  - Call gates (Type=12)
  - Trap gates (Type=14)
  - Interrupt gates (Type=15)
  - Task gates (Type=5)

Be very different from other descriptors, segmet selector instead of base address

Call-Gate Descriptor



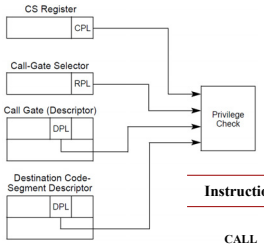
Protection

- Privilege Check for Control Transfer with Call Gate
- Example of Accessing Call Gates At Various Privilege Levels

Call-Gate Descriptor

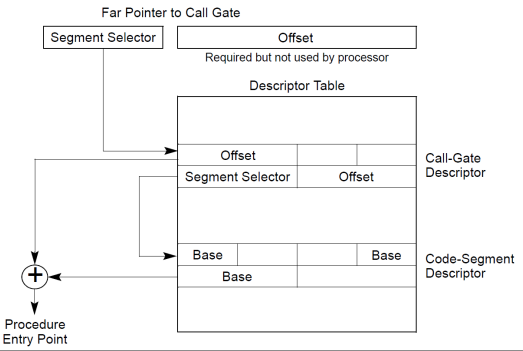
- It performs six functions
  - It specifies the **code segment** to be accessed
  - It defines **an entry point for a procedure** in the specified code segment
    - The **offset field** specifies the entry point
  - It specifies the **privilege level** required for a caller trying to access the procedure
  - If **a stack switch occurs**, it specifies the **number of optional parameters** to be copied between stacks
  - It defines the **size of values** to be pushed onto the target stack
    - 16-bit gates force 16-bit pushes and 32-bit gates force 32-bit pushes
  - It specifies whether **the call-gate descriptor is valid**

Privilege Check for Control Transfer with Call Gate

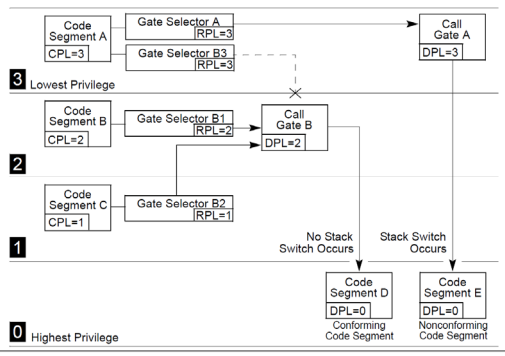


Instruction	Privilege Check Rules
CALL	CPL ≤ call gate DPL; RPL ≤ call gate DPL
	Destination **conforming** code segment DPL ≤ CPL
	Destination **nonconforming** code segment DPL ≤ CPL
JMP	CPL ≤ call gate DPL; RPL ≤ call gate DPL
	Destination **conforming** code segment DPL ≤ CPL
	Destination **nonconforming** code segment DPL = CPL

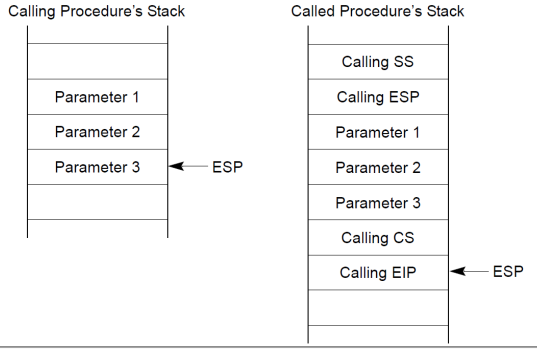
Call-Gate Mechanism

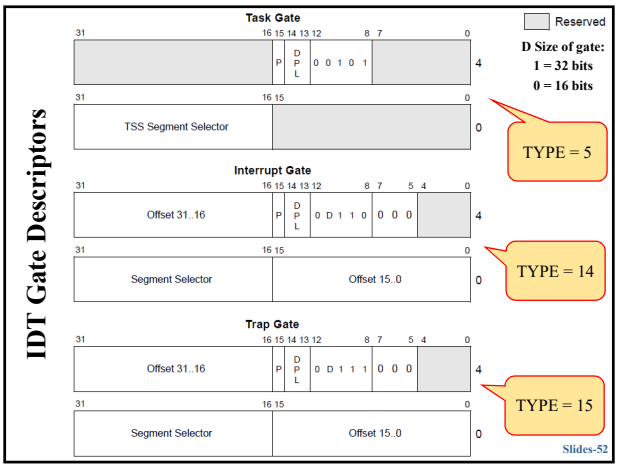


Example of Accessing Call Gates At Various Privilege Levels



Stack Switching

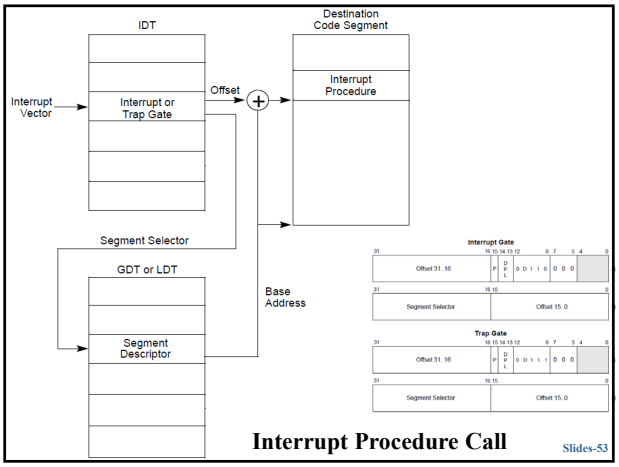




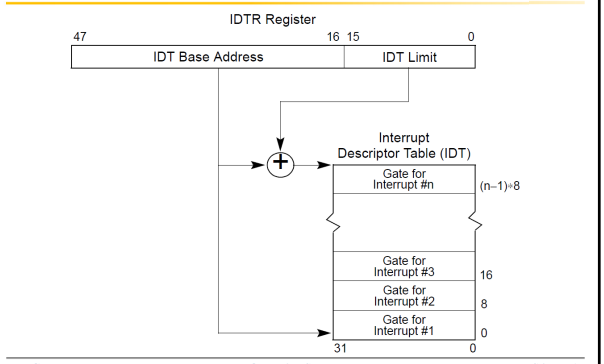
Interrupt And Exception Handling

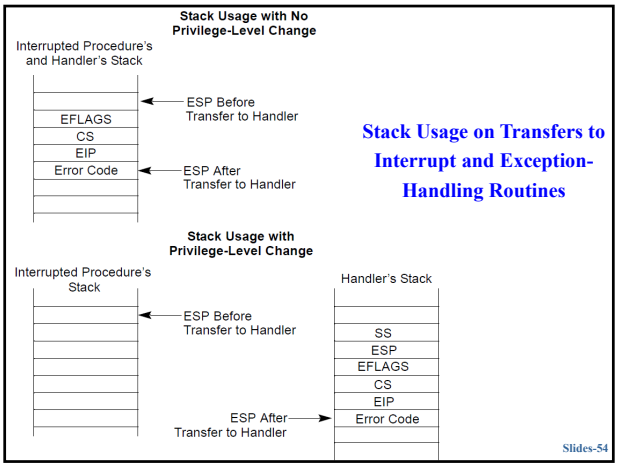
6

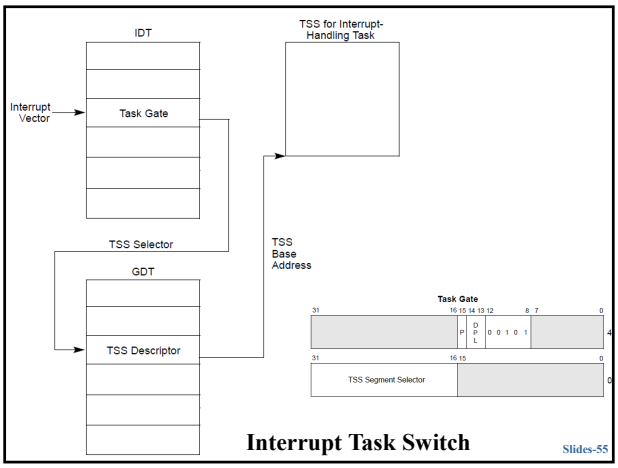
- Relationship of the IDTR and IDT
- IDT Gate Descriptors
- Interrupt Procedure Call
- Stack Usage on Transfers to Interrupt and Exception-Handling Routines
- Interrupt Task Switch



Relationship of IDTR and IDT







Interrupt Task Switch

Task State

- **Items define the state of the currently executing task**
  - The task's current execution space, defined by the segment selectors in the **segment registers** (CS, DS, SS, ES,FS, and GS)
  - The state of the **general-purpose registers**
  - The state of the **EFLAGS register**
  - The state of the **EIP register**
  - The state of **control register CR3**
  - The state of the **task register**
  - The state of the **LDTR register**
  - The **I/O map** base address and I/O map in TSS
  - **Stack pointers** to the privilege 0, 1, and 2 stacks in TSS
  - **Link to previously executed task** in TSS

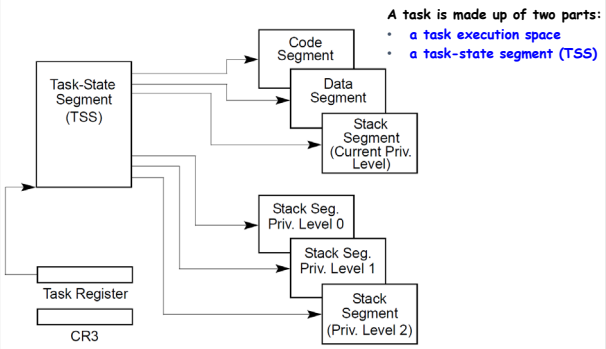
Task Management

- **Structure of a Task**
- **Task State**
- **Executing a Task**
- **Task Management Data Structures**
- **32-Bit Task-State Segment (TSS)**
- **TSS Descriptor**
- **Task Register**
- **Task Gates Referencing the Same Task**
- **Task Switching**
- **Nested Tasks**
- **Task Logical Address Space**

Executing a Task

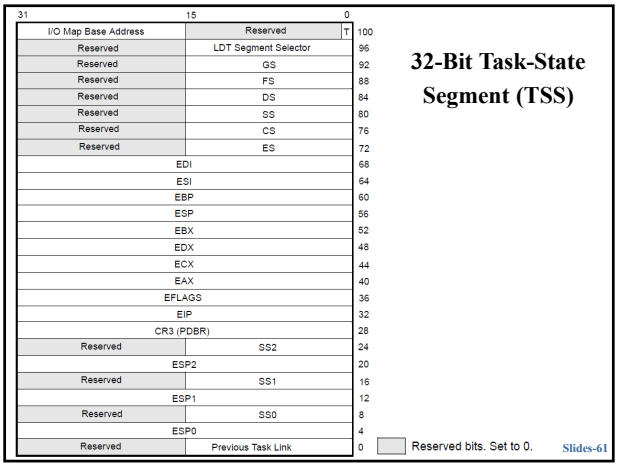
- **Software or the processor can dispatch a task for execution in one of the following ways:**
  - A **explicit call** to a task with the CALL instruction
  - A **explicit jump** to a task with the JMP instruction
  - An **implicit call** (by the **processor**) to an interrupt-handler task
  - An **implicit call** to an exception-handler task
  - **A return (initiated with an IRET instruction)**
    - when the NT flag in the EFLAGS register is set

Structure of a Task



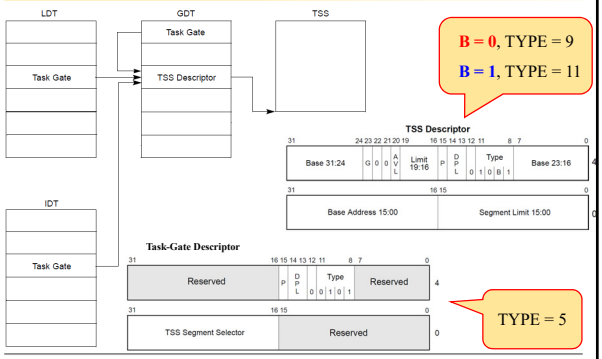
Task Management Data Structures

- **Task-state segment (TSS)**
- **Task-gate descriptor**
- **TSS descriptor**
- **Task register**
- **NT flag in the EFLAGS register**

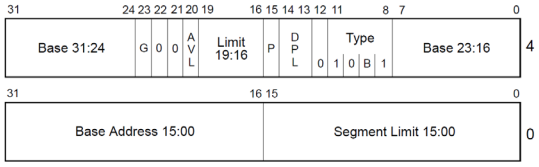


32-Bit Task-State Segment (TSS)

Task Gates Referencing the Same Task



TSS Descriptor



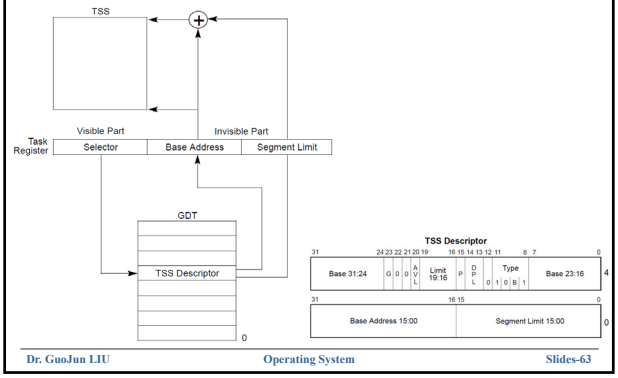
AVL	Available for use by system software
B	Busy flag
BASE	Segment Base Address
DPL	Descriptor Privilege Level
G	Granularity
LIMIT	Segment Limit
P	Segment Present
TYPE	Segment Type

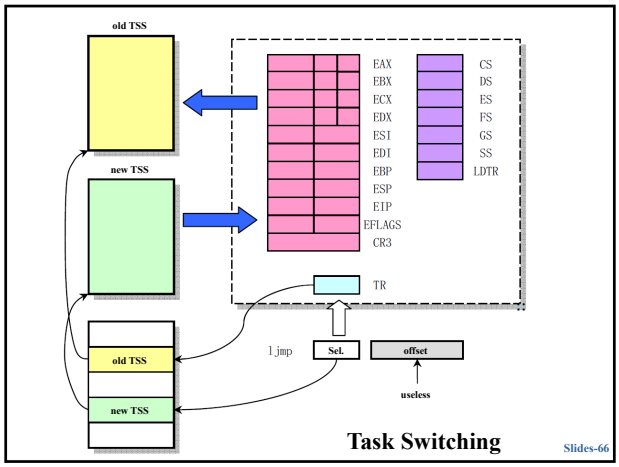
B = 0, TYPE = 9
B = 1, TYPE = 11

Task Switching

- **The processor transfers execution to another task in one of four cases**
  - The current program, task, or procedure executes a JMP or CALL instruction to a **TSS descriptor** in the GDT
  - The current program, task, or procedure executes a JMP or CALL instruction to **a task-gate descriptor** in the **GDT** or the **current LDT**
  - An interrupt or exception vector points to **a task-gate descriptor** in the IDT
  - The **current task executes an IRET** when the **NT flag** in the **EFLAGS register is set**

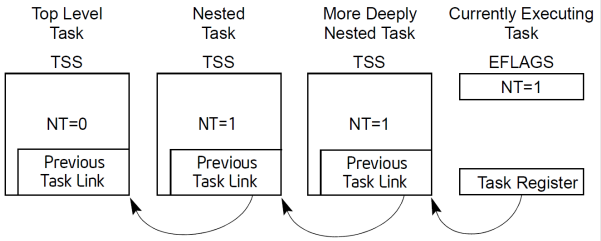
Task Register





Task Switching

Nested Tasks



Task Logical Address Space

- Through the segment descriptors in the GDT
- Through a shared LDT
- Through segment descriptors in distinct LDTs that are mapped to common addresses in linear address space

Summary

- Basic
- System Architecture Overview
- Protected-mode Memory Management
- Segment Descriptor
- Protection
- Interrupt And Exception Handling
- Task Management