Tutorial 11
Interrupts

Interrupts and Exceptions

What is it?

"A signal from hardware or software, such as a keystroke, that demands immediate attention and takes priority over other operations"

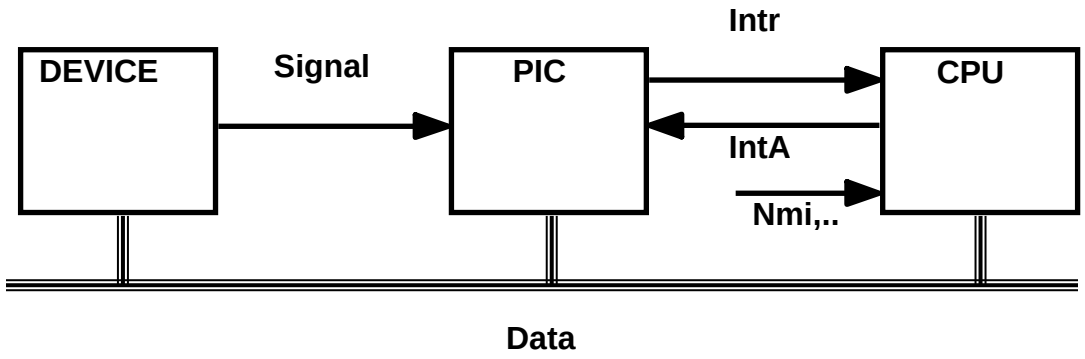
- Analogy:
  - Alarm clock
  - Pain

What Causes It?

- Interrupts
  - External to the CPU
  - Peripheral: I/O: mouse move/click, keyboard, timer
- Exceptions
  - Internal to the CPU
  - Exceptions: Processor detected:
    - **Fault:** restart from the causing instruction (e.g., page fault)
    - **Trap:** restart from the next instruction (e.g., overflow)
    - **Abort:** cannot restart (double fault)
- Programmed Exceptions ("software interrupts"):
  - e.g.: INT3, INT21, INT n
- The terms "interrupts" & "exceptions" are frequently intermixed

How Does It Work?

- Internal interrupts: handled completely by CPU + SW
- External interrupt involves CPU + SW + *peripherals*
  - External device delivers “INTR” signal (or NMI, SMI)
  - CPU acknowledges with “INTA” bus cycle
  - Device sends interrupt# on Data bus
  - Usually requires a PIC (Programmable Interrupt Controller) in system



Additional Players

- Flags
  - **Interrupt Flag** EFLAGS.IF flag: masks external interrupts
    - Set/Clear by STI/CLI instructions
  - **Trap Flag** EFLAGS.TF flag: is the CPU in TRAP mode (single-step)
    - Set/Clear by EFLAGS manipulation
  - **Resume Flag** EFLAGS.RF flag: when set disables debug-exception
- Instruction
  - **IRET**: "return from interrupt" = RET + few more things
  - Software interrupts: **INT n, INT3, INTO (4), BOUND (interrupt 5)**
- Signals (for external interrupts)
  - INTR: interrupt request
  - INTA: interrupt acknowledge (bus cycle) = IO#*C*R#
  - Data Bus: pass interrupt #
  - NMI, SMI: additional interrupt lines
    - NMI - non maskable (timer, power fail)
    - SMI - system management (OS independent power management)

Interrupt Handling

- Hardware side
  - Ensure transparency to the affected task
    - Will return to the point of interruption w/ the right flags
    - Does the minimum - only what software cannot do!
- Interrupt handlers
  - Transparent: preserve all used registers!
  - Short as possible. If cannot - enable interrupts during handling
    - Do not disable interrupts for a long time!
  - Avoid nesting of same interrupts, or make sure the software can handle that!
  - Some of the information can/should be extracted from the Stack
    (old CS:EIP, old SS:ESP, EFLAGS, error code)
- Interrupt handler writers must fully understand the relevant HW interaction (PIC + actual peripheral)

Exception and Interrupt Vectors

- Each exception and interrupt is associated with an identification number, called a **vector**.
  - Vectors 0-31 are assigned to the exceptions and NMI interrupt
    - 9,15,20-31 – Intel Reserved
  - Vectors 32-255 are designated and user defined interrupts.

- Exception Classification
  - **Fault**
    - An exception that can generally be corrected and then, once corrected, allows the program to be restarted with no loss of continuity
  - **Trap**
    - An exception that is reported immediately following the execution of the trapping instruction. Traps allow execution of a program or task to be continued without loss of program continuity
  - **Abort**
    - An exception that does not always report the precise location of the instruction causing the execution and does not allow restart of the program or task that caused the execution

Protected-Mode Exceptions

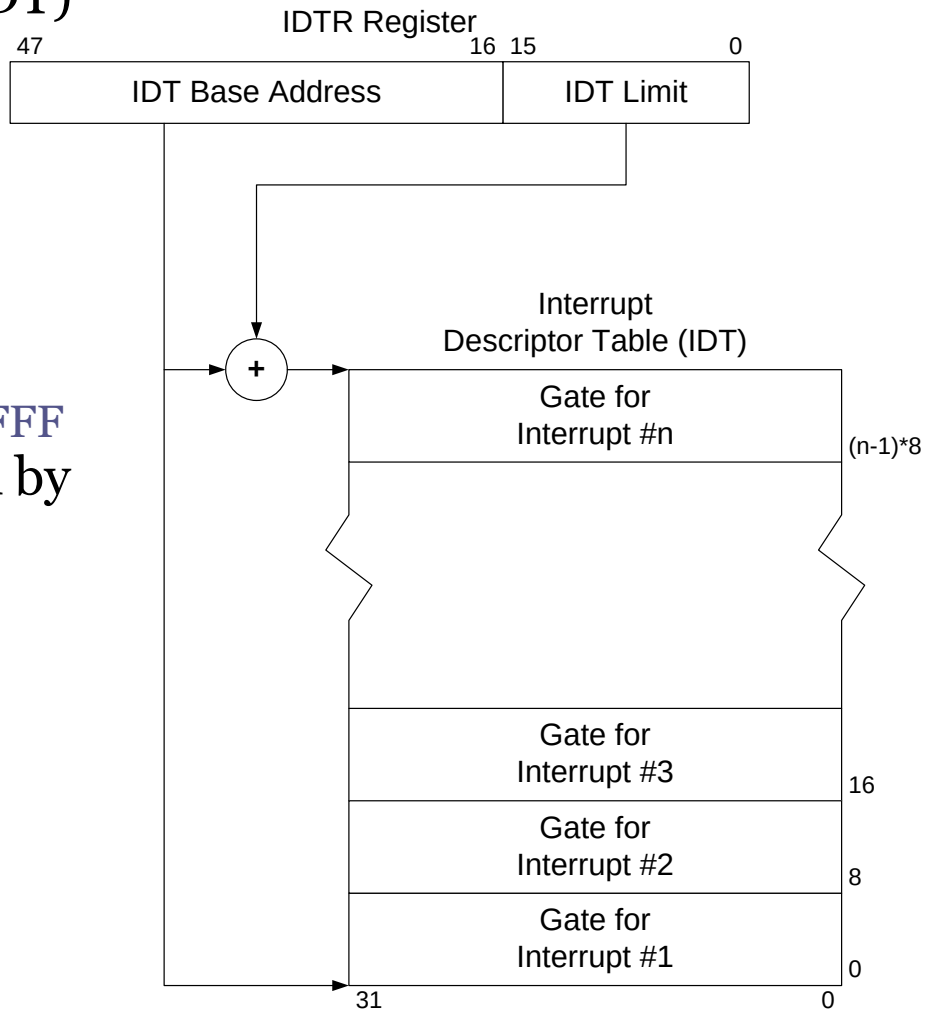
No.	Mne-monic	Description	Type	Error Code	Source
0	#DE	Divide by Zero	Fault	No	DIV / IDIV instructions
1	#DB	Debug	Fault/Trap	No	DR conditions / INT1 instruction.
2	-	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt
3	#BP	Breakpoint	Trap	No	INT3 instruction
4	#OF	Overflow	Trap	No	INTO instruction
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction
6	#UD	Invalid Opcode	Fault	No	UD2 instruction / reserved opcode
7	#NM	Device Not available	Fault	No	FP or WAIT/FWAIT instructions
8	#DF	Double Fault	Abort	Yes (Zero)	Any exception, interrupt, NMI
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access
11	#NP	Segment Not Present	Fault	Yes	Loading SR / accessing system segment
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations / SS register loads
13	#GP	General Protection Fault	Fault	Yes	Memory references / protection checks
14	#PF	Page Fault	Fault	Yes	Any memory reference
16	#MF	Floating-Point Error	Fault	No	FP or WAIT/FWAIT instructions
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory
18	#MC	Machine Check	Abort	No	Model dependent
19	#XF	Streaming SIMD Extensions	Fault	No	SIMD floating-point instructions

Exceptions and Interrupts Priority

Priority	Descriptions
1 (Highest)	Hardware Reset and Machine Checks
2	Trap on Task Switch
3	External Hardware Interventions -FLUSH, STOPCLK, SMI, INIT
4	Traps on the Previous Instruction - Breakpoints, Debug Traps
5	External Interrupts - NMI, Maskable Hardware Interrupts
6	Faults from Fetching Next Instruction - Code breakpoint, code-segment violation, Code page fault
7	Faults from Decoding the Next Instruction - Instruction length > 15, illegal Opcode
8 (Lowest)	Faults on Executing an Instruction

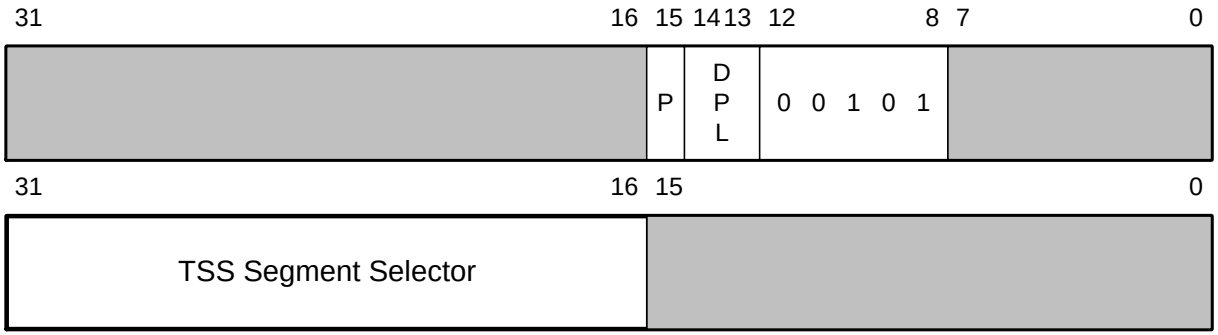
Interrupt Descriptor Table

- Interrupt Descriptor Table (IDT)
  - Located in virtual memory
  - Each entry contains 8-bytes Interrupt Descriptor
- Interrupt Descriptor Table Register (IDTR)
  - Contains:
    - IDT Linear Base Address
    - IDT Limit in bytes, max 0xFFFF
- Operating System loads IDTR by executing LIDT instruction
  - LIDT is a privilege instruction
  - Can be executed only in ring0
- One can observe IDTR by performing SIDT instruction
  - SIDT can be executed in any privilege level

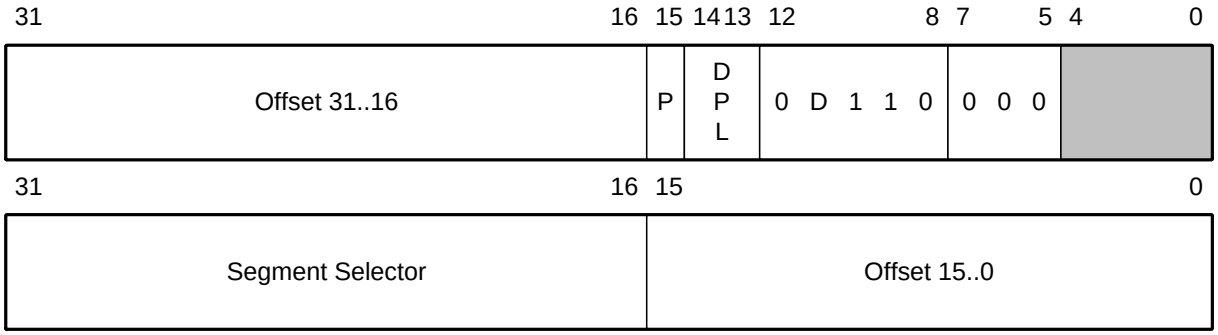


IDT Descriptors

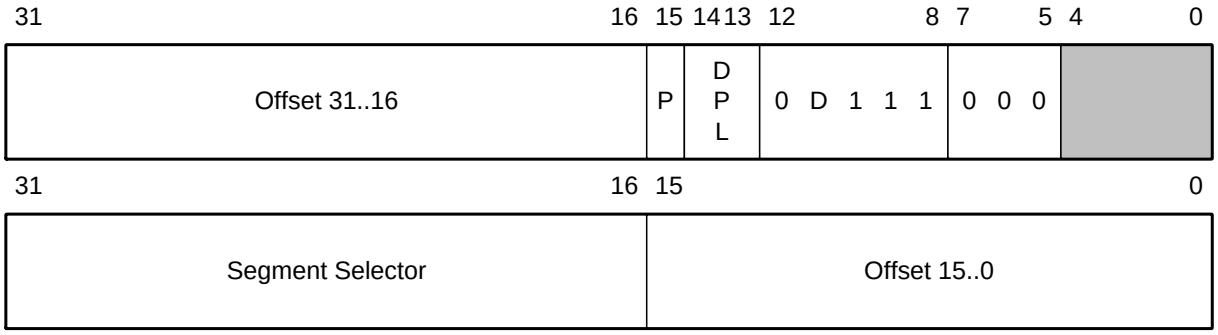
Task gate



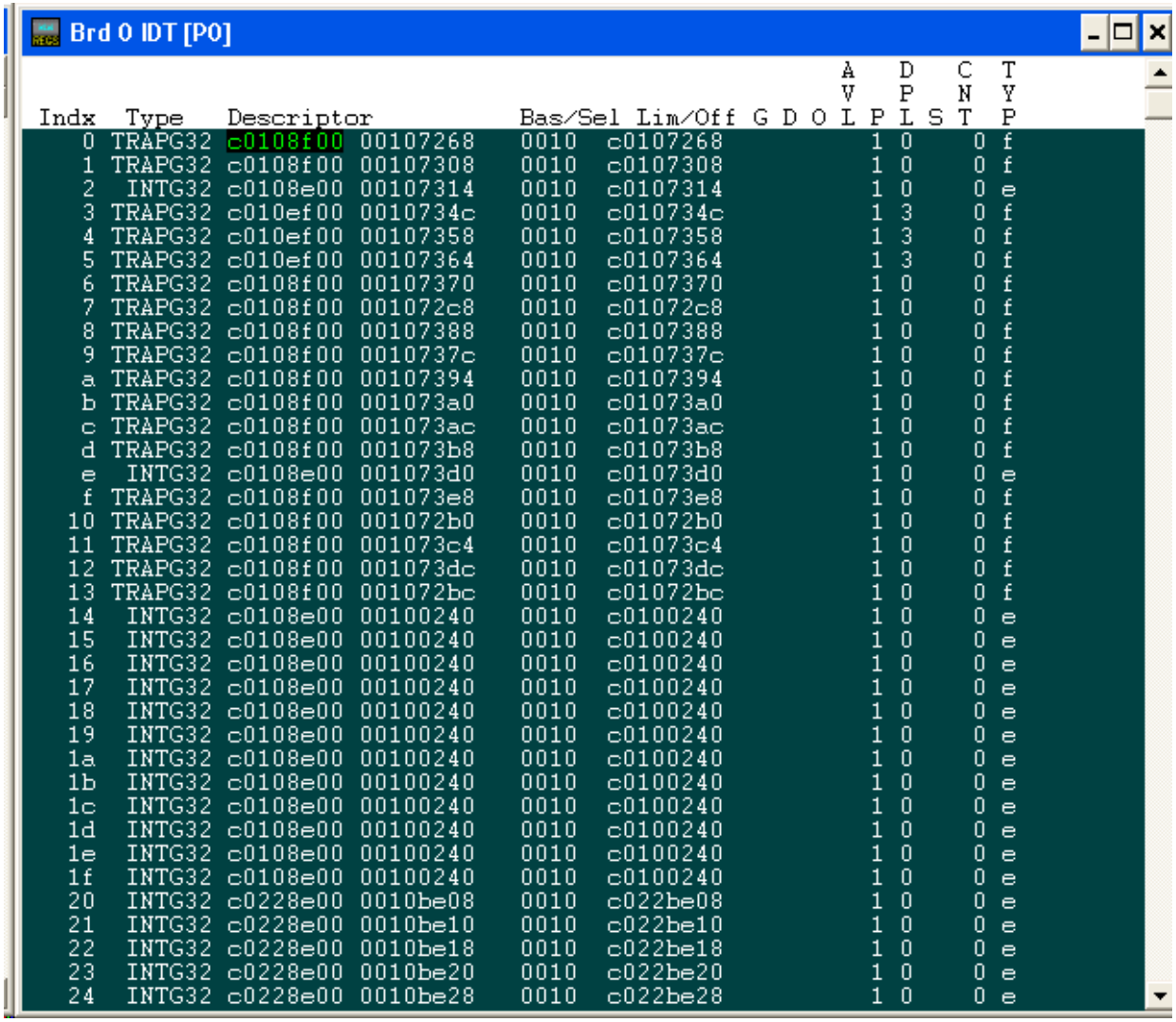
Interrupt Gate



Trap gate



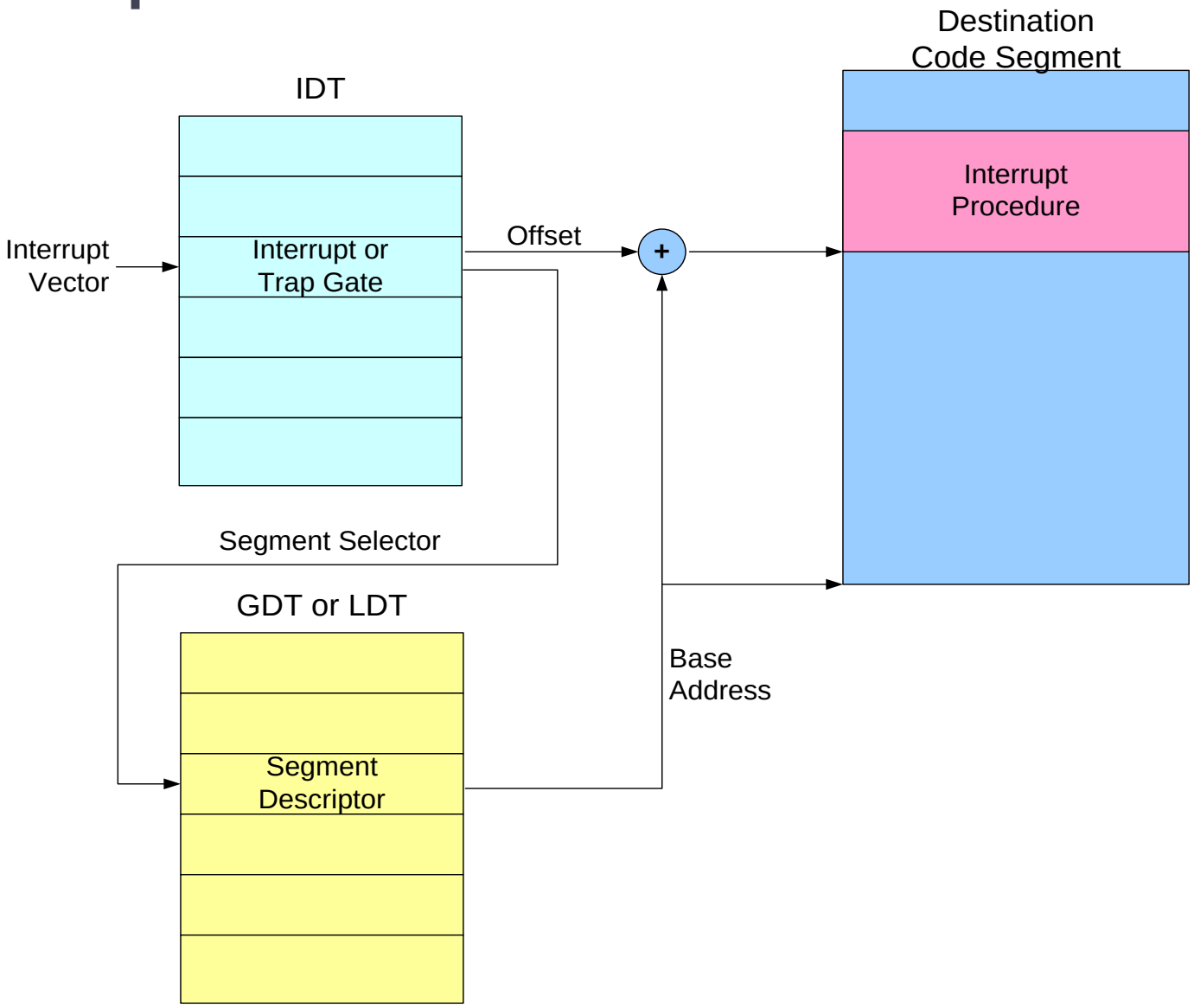
Linux IDT - Example



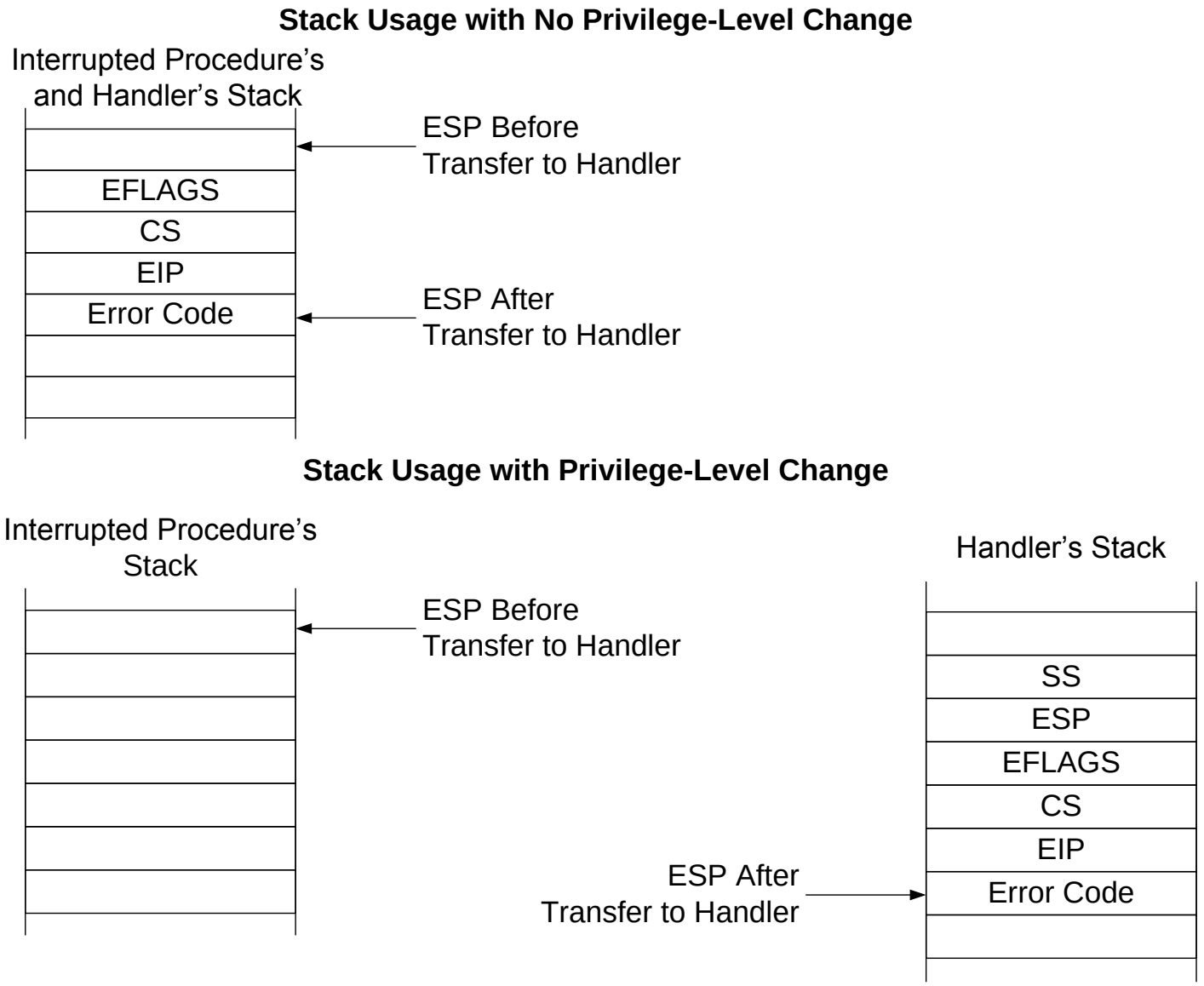
Brd 0 IDT [P0]

Indx	Type	Descriptor		Bas/Sel	Lim/Off	G	D	O	AVL	DPL	S	CNT	TYP
0	TRAPG32	c0108f00	00107268	0010	c0107268				1	0		0	f
1	TRAPG32	c0108f00	00107308	0010	c0107308				1	0		0	f
2	INTG32	c0108e00	00107314	0010	c0107314				1	0		0	e
3	TRAPG32	c010ef00	0010734c	0010	c010734c				1	3		0	f
4	TRAPG32	c010ef00	00107358	0010	c0107358				1	3		0	f
5	TRAPG32	c010ef00	00107364	0010	c0107364				1	3		0	f
6	TRAPG32	c0108f00	00107370	0010	c0107370				1	0		0	f
7	TRAPG32	c0108f00	001072c8	0010	c01072c8				1	0		0	f
8	TRAPG32	c0108f00	00107388	0010	c0107388				1	0		0	f
9	TRAPG32	c0108f00	0010737c	0010	c010737c				1	0		0	f
a	TRAPG32	c0108f00	00107394	0010	c0107394				1	0		0	f
b	TRAPG32	c0108f00	001073a0	0010	c01073a0				1	0		0	f
c	TRAPG32	c0108f00	001073ac	0010	c01073ac				1	0		0	f
d	TRAPG32	c0108f00	001073b8	0010	c01073b8				1	0		0	f
e	INTG32	c0108e00	001073d0	0010	c01073d0				1	0		0	e
f	TRAPG32	c0108f00	001073e8	0010	c01073e8				1	0		0	f
10	TRAPG32	c0108f00	001072b0	0010	c01072b0				1	0		0	f
11	TRAPG32	c0108f00	001073c4	0010	c01073c4				1	0		0	f
12	TRAPG32	c0108f00	001073dc	0010	c01073dc				1	0		0	f
13	TRAPG32	c0108f00	001072bc	0010	c01072bc				1	0		0	f
14	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
15	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
16	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
17	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
18	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
19	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
1a	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
1b	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
1c	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
1d	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
1e	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
1f	INTG32	c0108e00	00100240	0010	c0100240				1	0		0	e
20	INTG32	c0228e00	0010be08	0010	c022be08				1	0		0	e
21	INTG32	c0228e00	0010be10	0010	c022be10				1	0		0	e
22	INTG32	c0228e00	0010be18	0010	c022be18				1	0		0	e
23	INTG32	c0228e00	0010be20	0010	c022be20				1	0		0	e
24	INTG32	c0228e00	0010be28	0010	c022be28				1	0		0	e

Interrupt Procedure Call

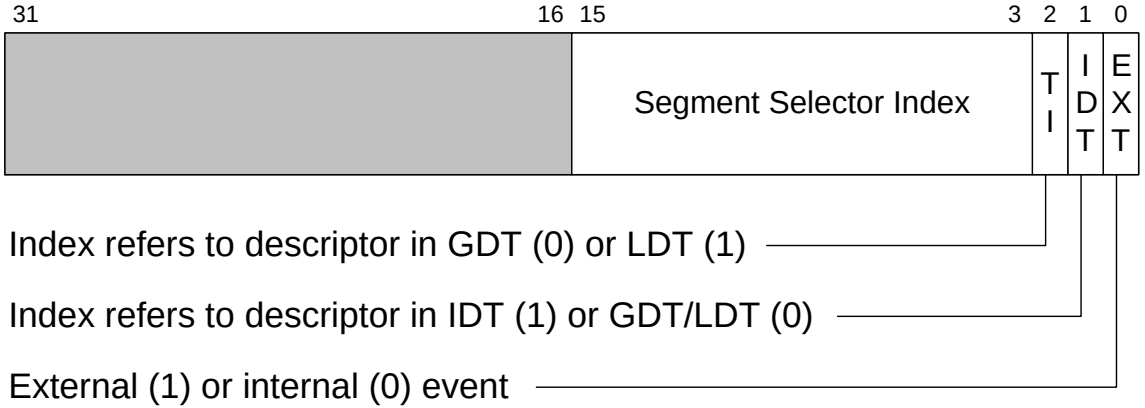


Stack Usage on Interrupt Transfer

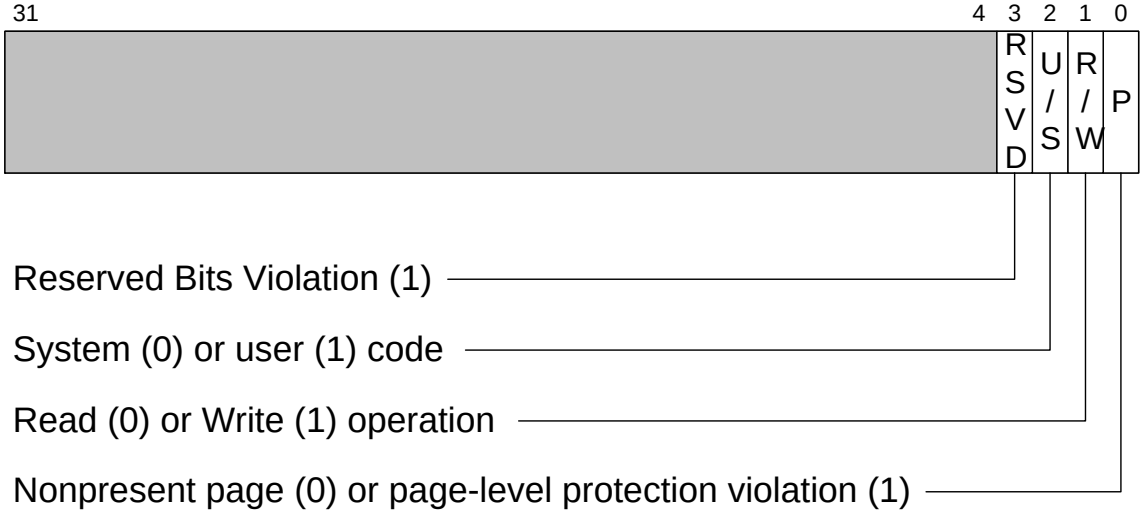


Exception Error Codes

General Error Code



Page-Fault Error Code



Interrupt and Exception Classes

Class	Vector Number	Description
Benign Exception and Interrupts	1	Debug Exception
	2	NMI Interrupt
	3	Breakpoint
	4	Overflow
	5	BOUND Range Exceeded
	6	Invalid Opcode
	7	Device Not Available
	9	Coprocessor Segment Overrun
	16	Floating-Point Error
	17	Alignment Check
	18	Machine Check
	19	SIMD floating-point extensions
	All	Software Interrupts INT n
	All	External Interrupts INTR
Contributory Exceptions	0	Divide Error
	10	Invalid TSS
	11	Segment Not Present
	12	Stack Fault
	13	General Protection
Page Faults	14	Page Fault

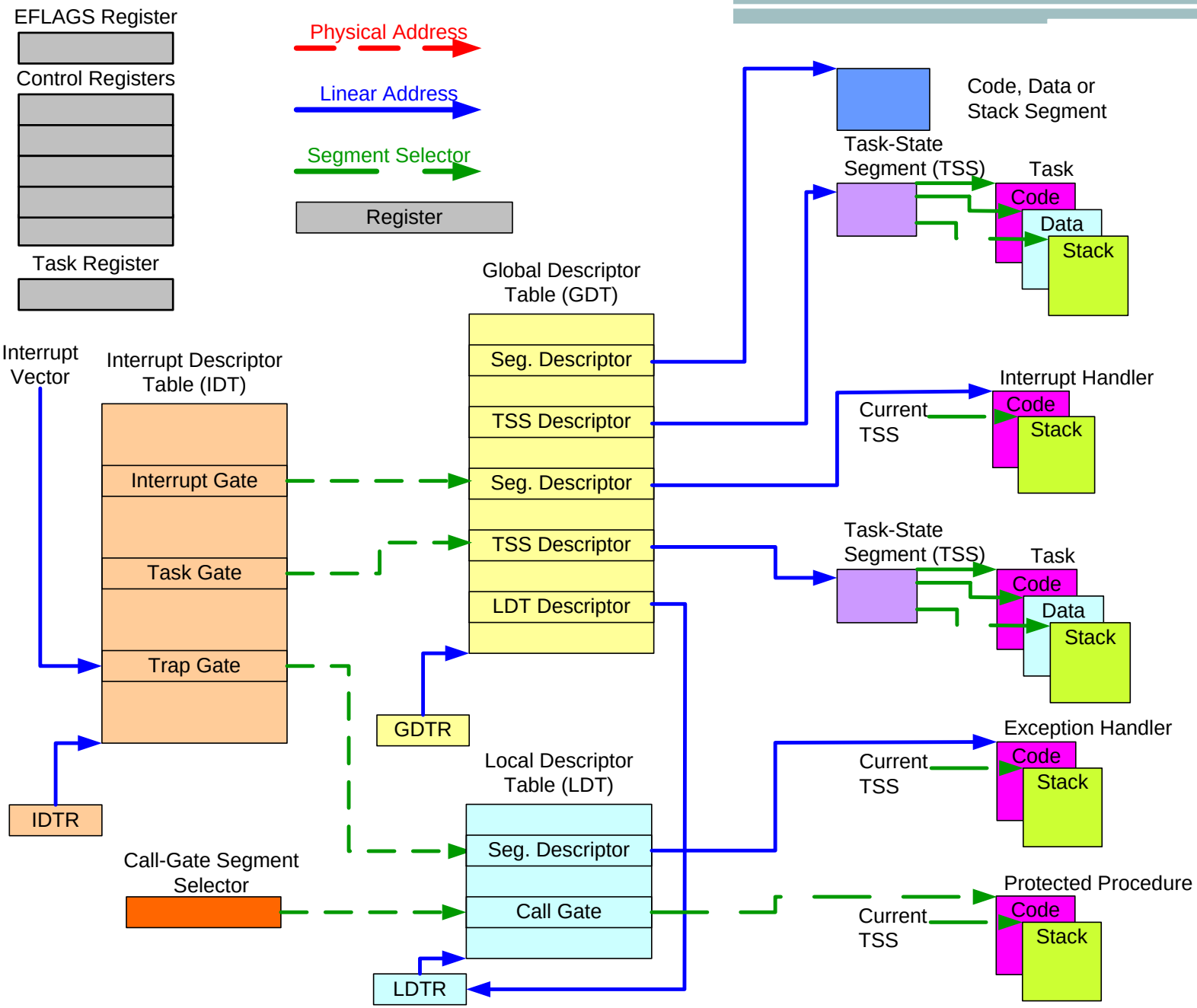
Conditions for Generating a Double Fault

First Exception	Second Exception		
	Benign	**Contributory**	**Page Fault**
Benign	Handle Exceptions Serially	Handle Exceptions Serially	Handle Exceptions Serially
Contributory	Handle Exceptions Serially	Generate a Double Fault	Handle Exceptions Serially
Page Fault	Handle Exceptions Serially	Generate a Double Fault	Generate a Double Fault

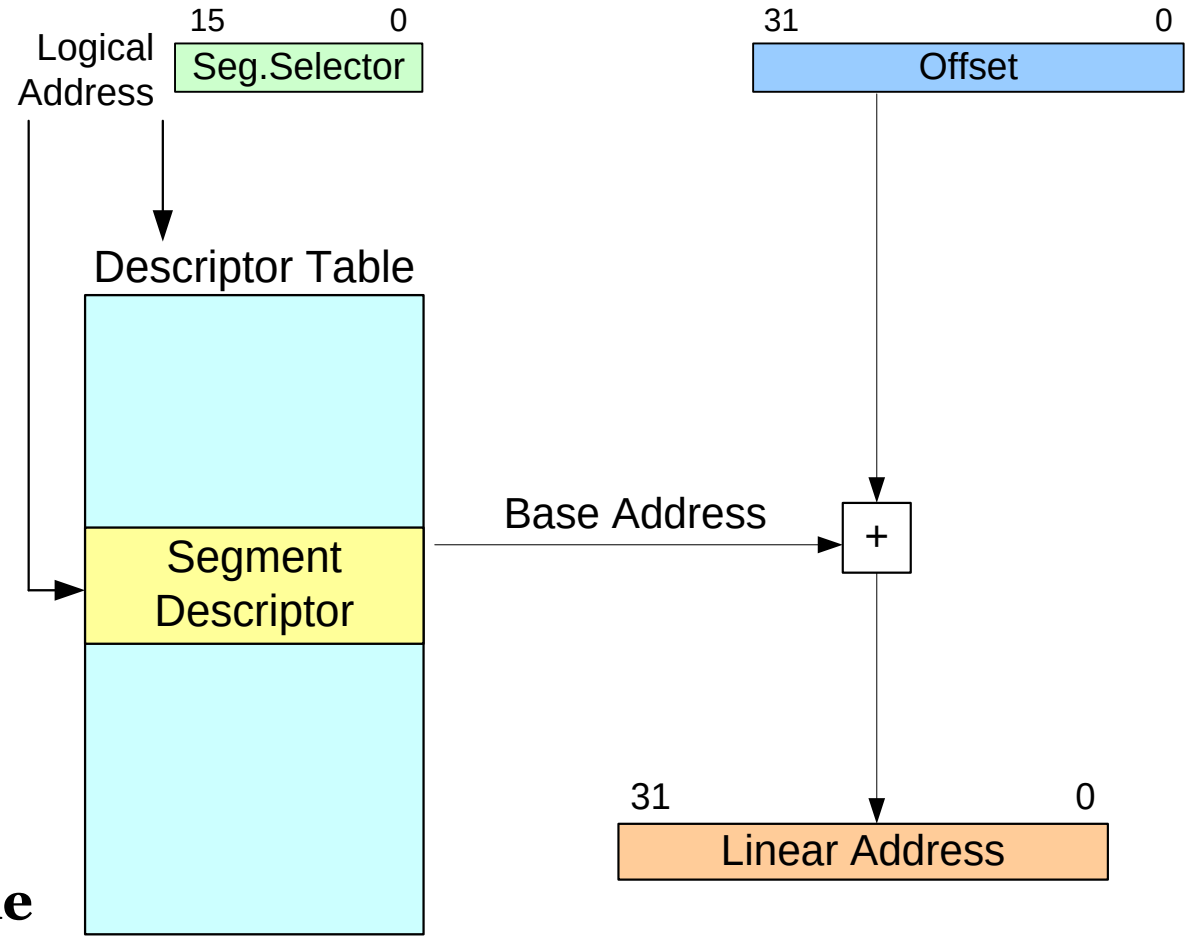
Triple Fault and Shutdown State

- **Triple fault** is a case when another exception occurs while attempting to call the double-fault handler, the processor enters **shutdown** mode.
- Shutdown mode is similar to the state following execution of an HLT instruction.
- In this mode the processor stops executing instructions until an NMI interrupt, SMI interrupt, hardware reset, or INIT# is received.

System Architecture Summary


EFLAGS Register
Control Registers
Task Register
Physical Address
Linear Address
Segment Selector
Register
Global Descriptor Table (GDT)
Seg. Descriptor
TSS Descriptor
Seg. Descriptor
TSS Descriptor
LDT Descriptor
Code, Data or Stack Segment
Task-State Segment (TSS)
Task
Code
Data
Stack
Interrupt Vector
Interrupt Descriptor Table (IDT)
Interrupt Gate
Task Gate
Trap Gate
Interrupt Handler
Current TSS
Code
Stack
Task-State Segment (TSS)
Task
Code
Data
Stack
GDTR
IDTR
Local Descriptor Table (LDT)
Exception Handler
Current TSS
Code
Stack
Call-Gate Segment Selector
Seg. Descriptor
Call Gate
Protected Procedure
Current TSS
Code
Stack
LDTR

Linear Address Computation

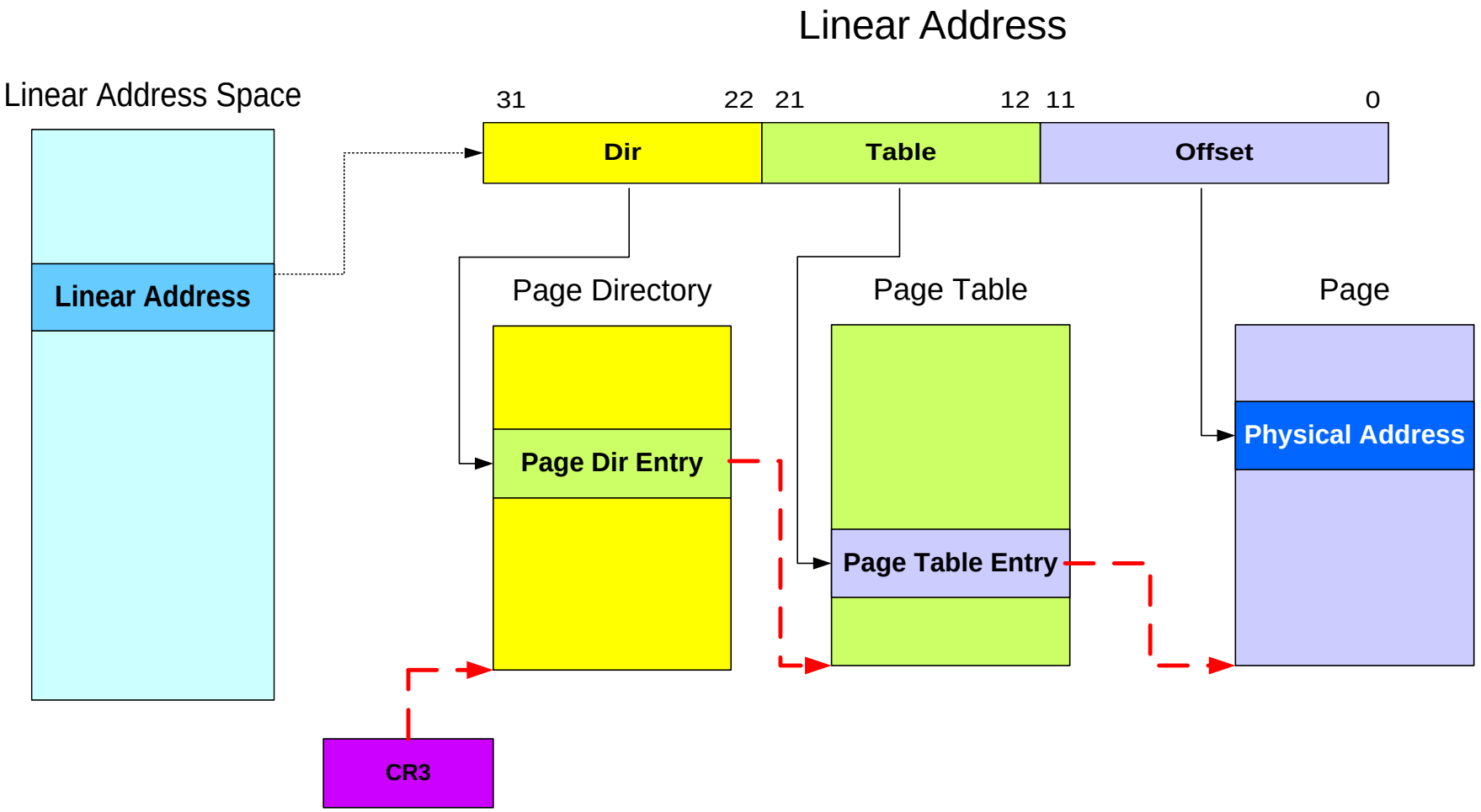


- **Protected Mode**

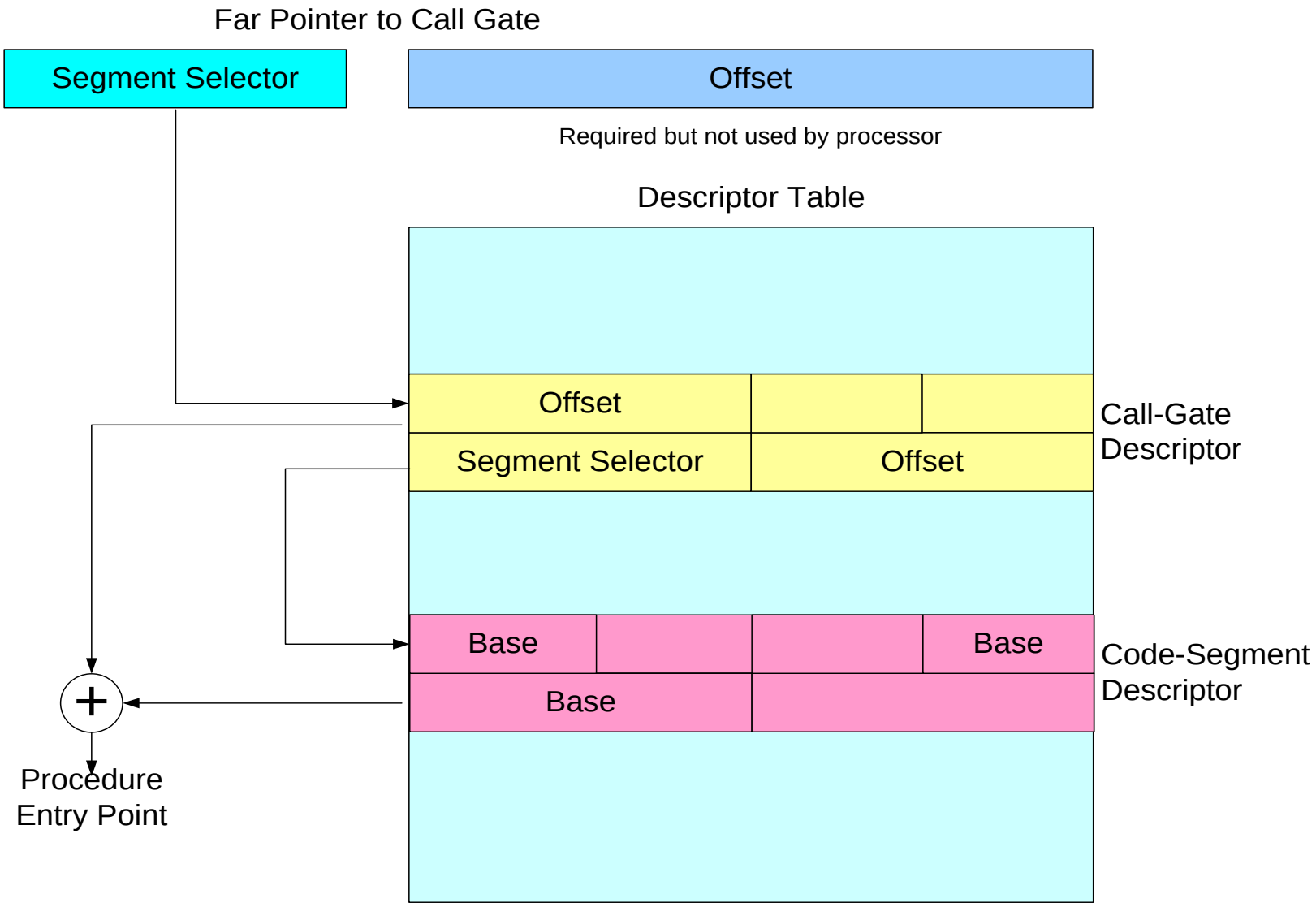
Linear address =

Descriptor_Table[f(segment)].base + offset

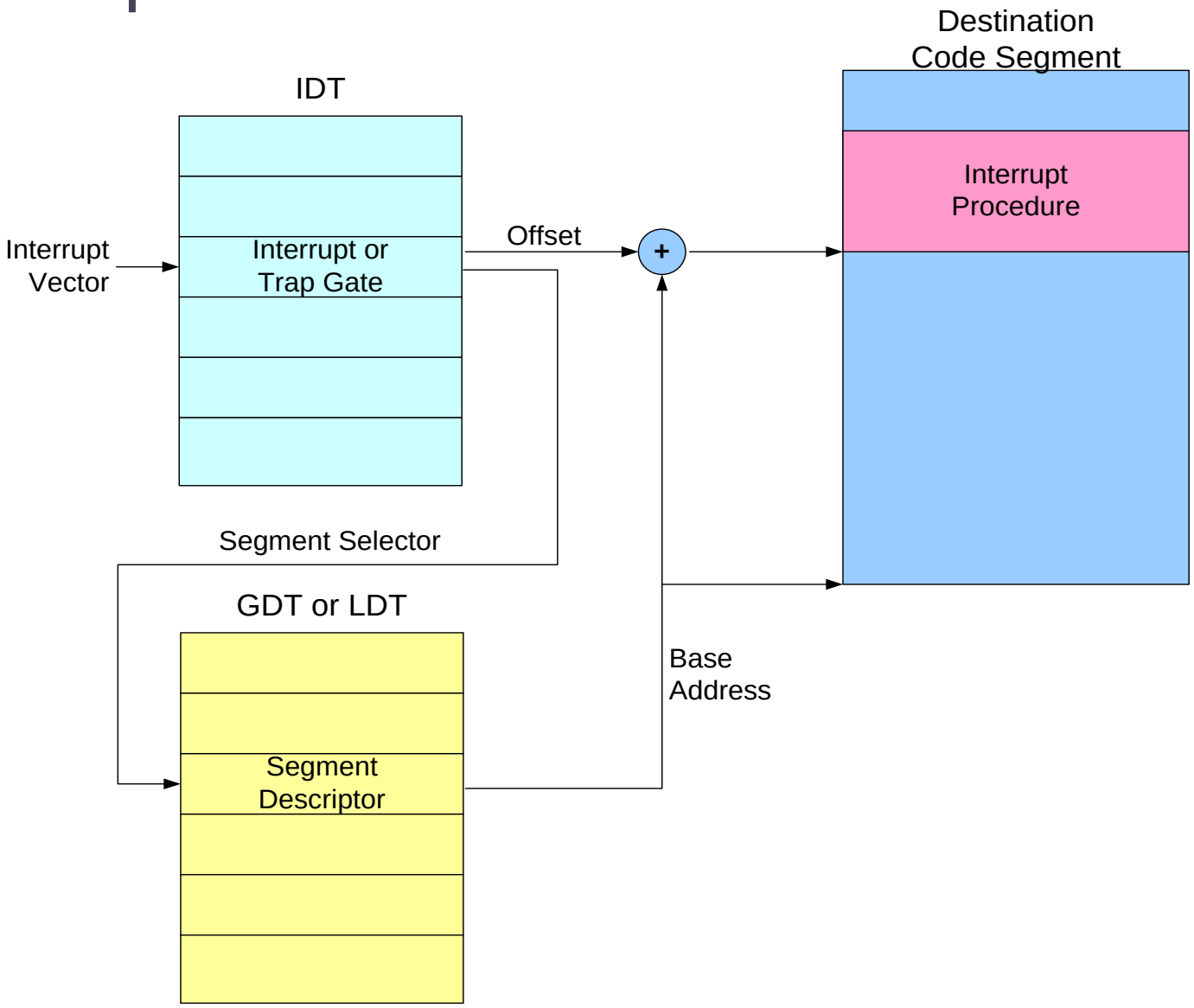
Page Translation Mechanism



Accessing a Code Segment Through a Call Gate

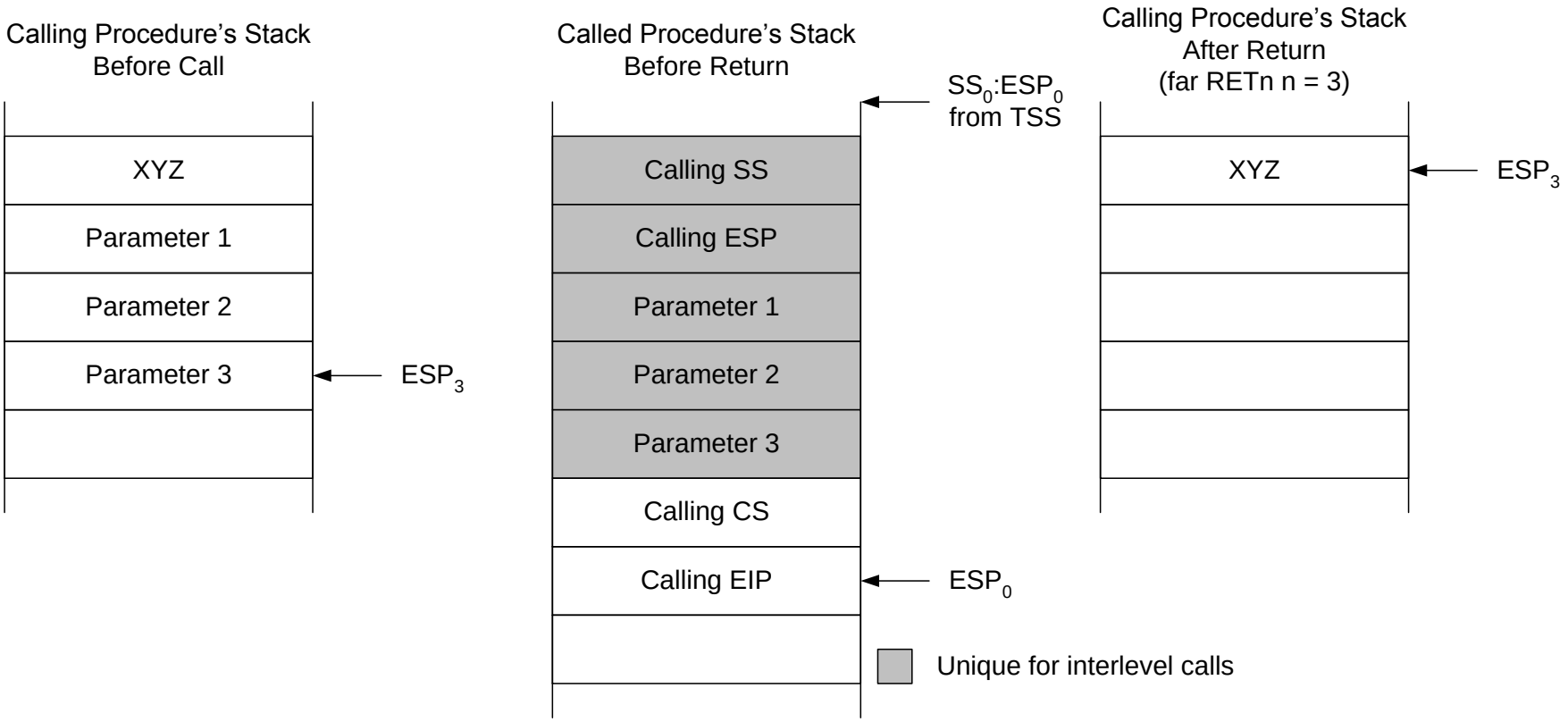


Interrupt Procedure Call



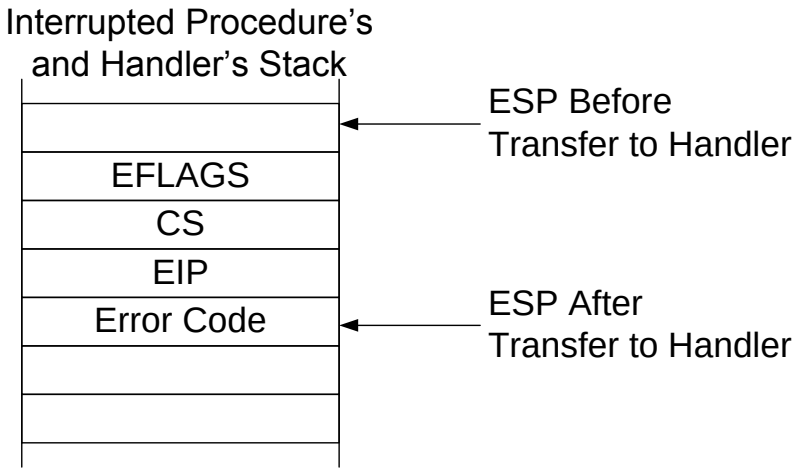
Inter Level Call - Stack Switching

- On inter-level call: CPL changes, Control transferred, Stack switched
- Stack Switch
  - New (privileged) SS:ESP is taken from the TSS
  - Old SS:ESP is stored on new stack
  - Parameters are copied to new stack



Stack Usage on Interrupt Transfer

Stack Usage with No Privilege-Level Change



Stack Usage with Privilege-Level Change