# Digital microsystems design

Marius Marcu

2019

# Objectives

Specific objectives

- Protected mode of operation of modern IA-32 processors architecture

# Outline

- Introduction
- Protected mode
  - Segmentation
  - Paging

# Introduction

IA-32 architecture supports several operating modes:

- Real mode
- Protected mode
- System management mode
- Virtual-8086 mode

# Introduction

Protected mode of modern microprocessors provide operatin
systems the hardware support needed to build safe and secu
multi-tasking and multi-threading environments

80386 implemented the reference support for OS kernel
implementation (i386) based on protected operation mode:

- Privilege levels
- Memory management: segmentation and paging

Modern IA-32 processors are backward compatible with 386,
using the same model for OS kernel implementation

# Introduction

IA-32 processors

- 32 address lines – 4 GB of physical memory
- 32 data lines

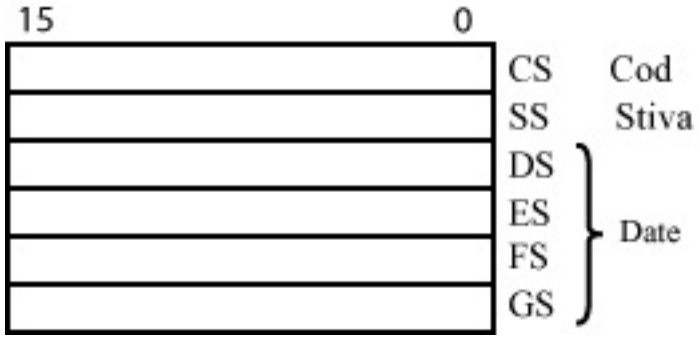
AMD-64 processors

- >32-64 address lines
- 64 data lines

# i386

## Registers set

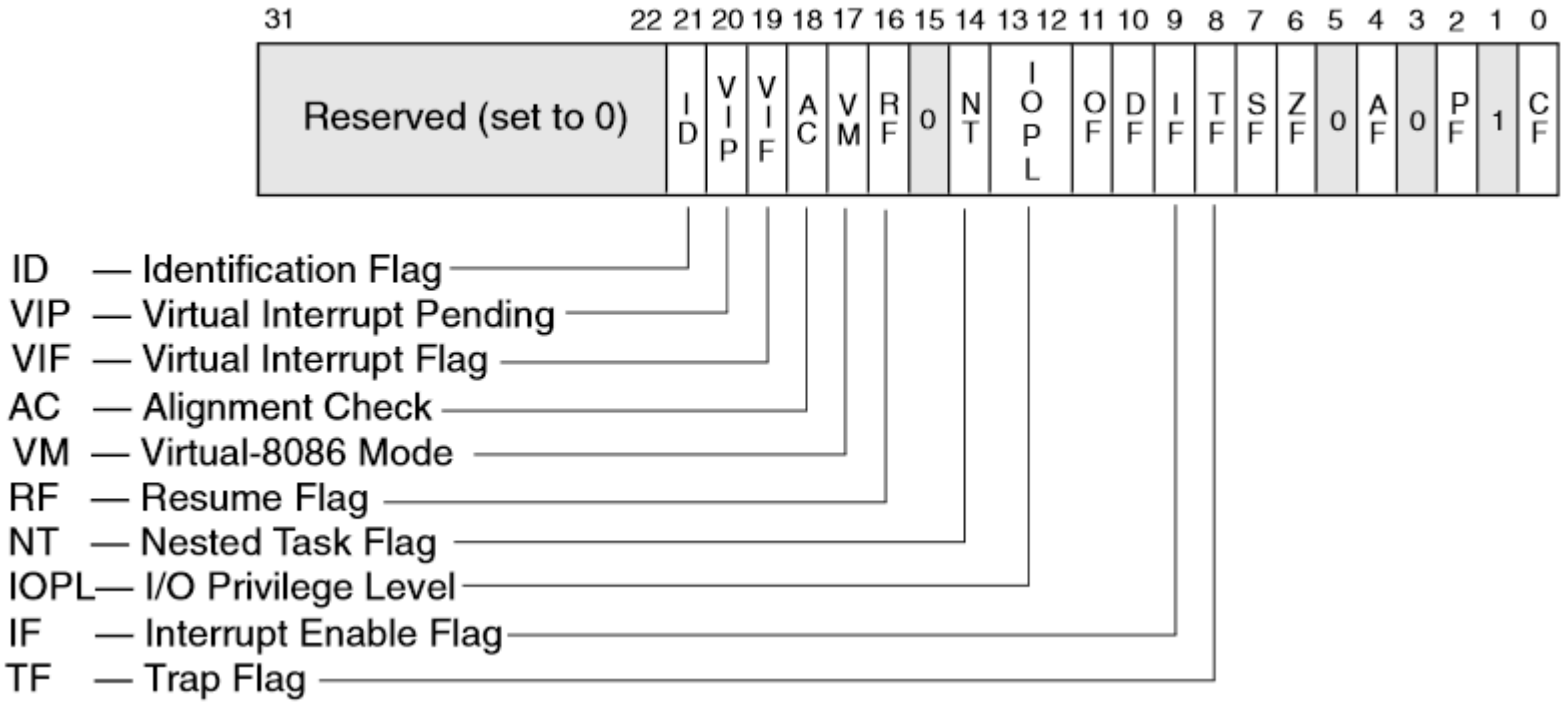
- Segment registers
  - 16 bits
  - Two new registers for data segments: FS and GS



# i386

## Registers set

– EFALGS



# i386

Real mode

- 8086 execution with
  - 32 bits registers support
  - protected mode access
- Memory segments of 64 KB
- Physical and logical addressing modes
  - Physical address = Segment * 16 + Offset
- Interrupts handling similar with 8086
- Non protected instructions

# i386

## Protected mode

– Which are the main limits and drawbacks of real mode of operation for x86 processors?

- Limitation of physical memory address space
- Task level data and code lack of security
- Task isolation

# i386

Protected mode

- IA-32 provide support for OS and system development software including features like
  - Memory management
  - Protection of software modules
  - Multitasking
  - Exception and interrupt handling
  - Multiprocessing
  - Cache management
  - Hardware resources and power management
  - Debugging and performance monitoring

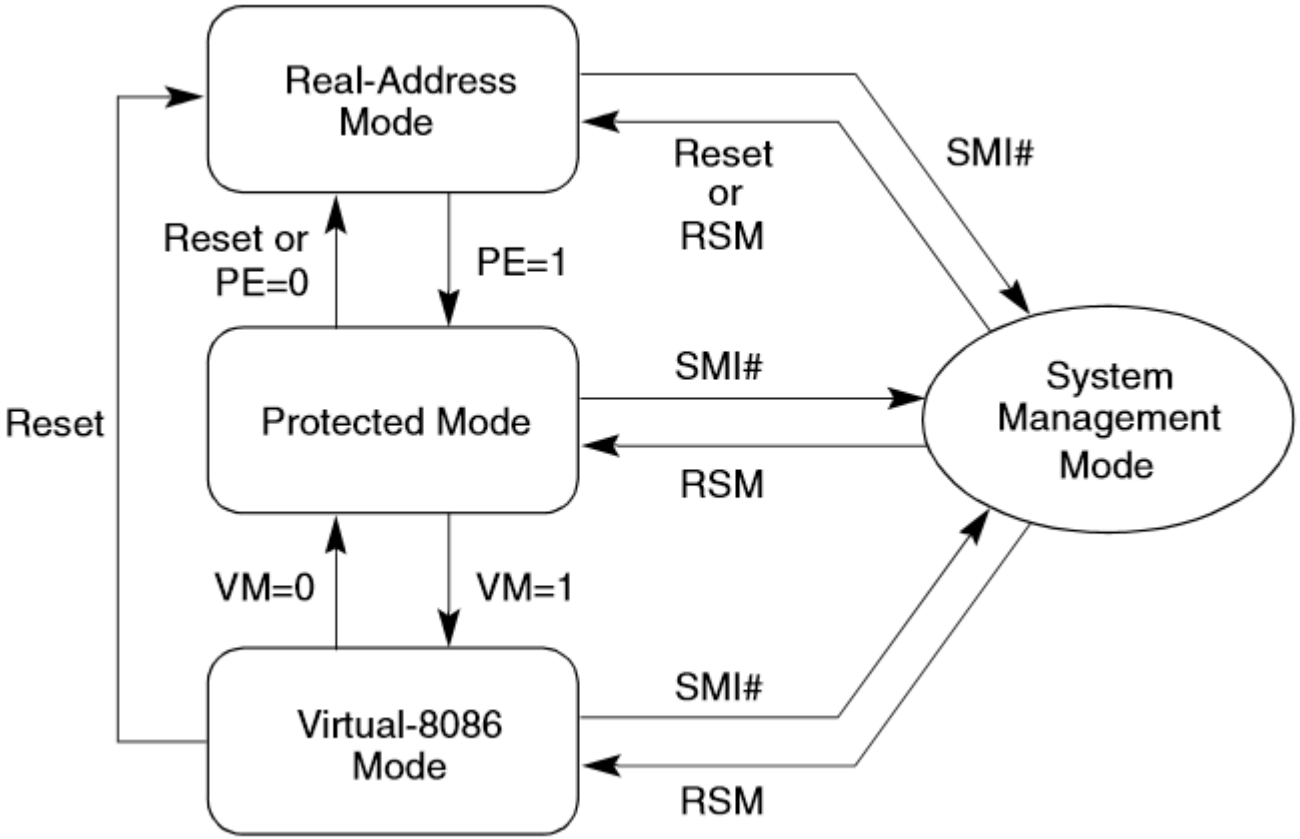
# i386

Protected mode

- This is the native operating mode of the i386 processors
- In this mode all instructions and architectural features are available providing the highest performance and capability
- This is the recommended mode for all new applications and operating systems
- Modern OS operates on top of this mode

# IA-32

Protected mode setup



# Protected mode

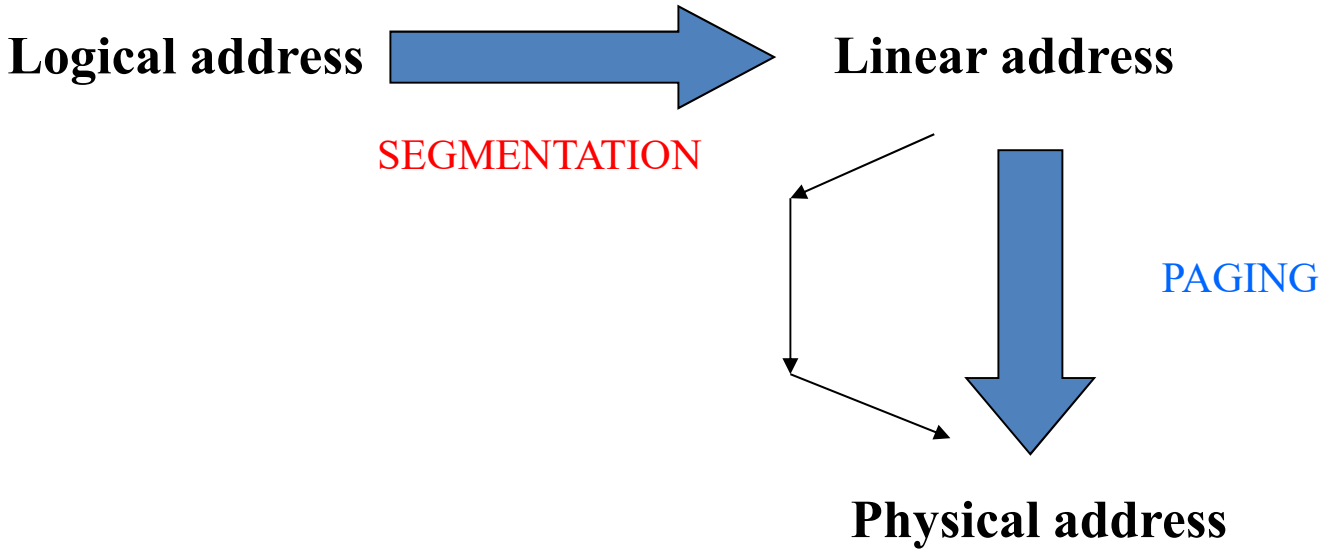
Memory management

- Segmentation – mechanism of isolating individual code, data or sta blocks so that multiple tasks can run on the same processor withou interfering one another – mandatory
- Paging – mechanism for implementing a conventional virtual memo system where sections of the task's memory are mapped into the physical memory as needed – optional

# Protected mode

Memory management

- Logical address (segment:offset)
- Linear address (0h – ffffffffh)
- Physical address



# Protected mode

Memory management – segmentation

- Provides a mechanism for dividing the processor's addressable memory space (called the linear address space) into smaller protected address spaces called segments.
- Segments hold
  - code, data or stack
  - system data structures (such as a TSS or LDT)
- In a multitasking environment, each task is assigned with its own se of segments

# Protected mode

Memory management – segmentation

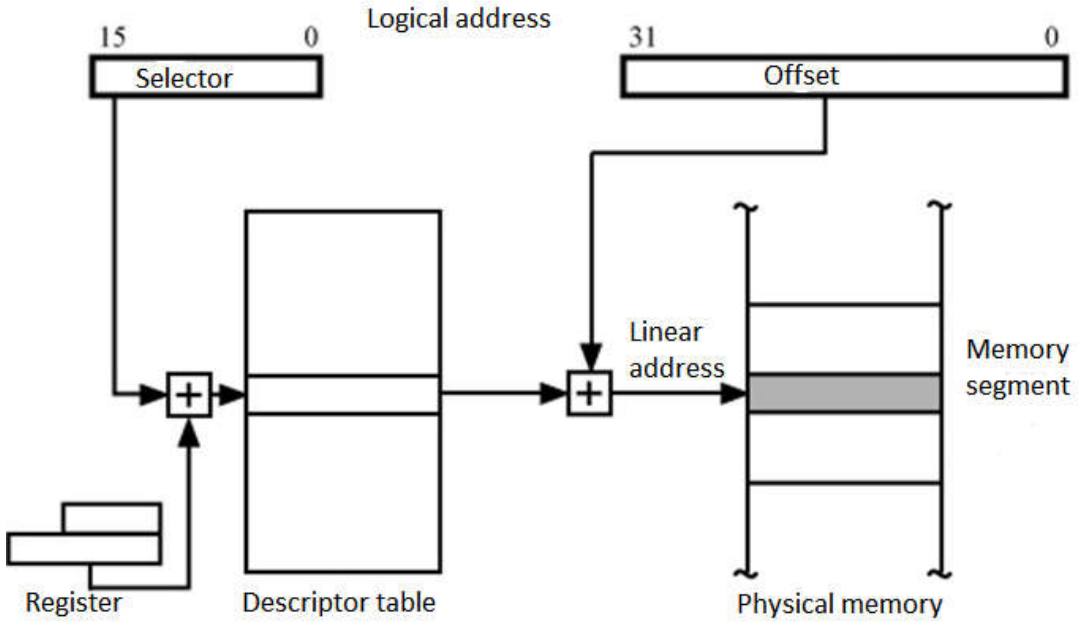
– The processor performs several validations at segment level

- Segment limits (variable size)
- Task level access (programs do not interfere each other)
- Segment types (operations allowed on values)

# Protected mode

## Memory management – segmentation

- Segment selector is a 16-bit identifier for a segment
  - It is an index pointing to a segment descriptors table location
  - It does not point directly to the segment, but instead points to the segment descriptor tha defines the segment



# Protected mode

## Memory management – segmentation

– Selector segment structure


- Index -13 bits
  - $2^{13}$ = 8192 – locations
- TI – table indicator
  - GDT – global descriptors table
  - LDT – local descriptors table
- One task can access max. 8192 shared segments and 8192 local segments

# Protected mode

Memory management – segmentation

- Segment registers are indexes to locations of one of the following tables: GDT, LDT
- If paging is not used, linear address space is mapped directly into th physical address space of the processor
- The physical address space of the processor is defined as the range addresses the processor can generate on its address bus

# Protected mode

Memory management – segmentation

- Descriptors tables
  - GDT – common to all tasks
  - LDT – local to a specific task
  - IDT – interrupt vector table (system level)
- Descriptors tables registers
  - Stores the addresses of the descriptors table in the memory
  - GDTR – GDT register
  - LDTR – LDT register
  - IDTR – IDT register

# Protected mode

Memory management – segmentation

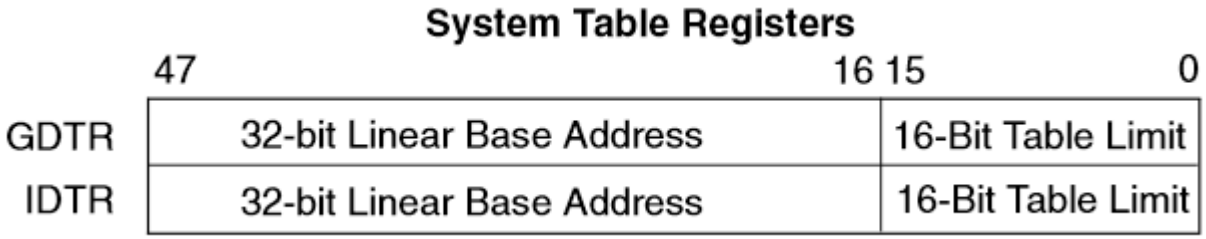
– Example – how many memory accesses are used when paging is no enabled

```
MOV DS:[EBX+2], AX
```

# Protected mode

Memory management – segmentation

– System table registers structure




– LDT are allocated per task

- They are defined as memory segments
- LDTR keeps LDT base address for the current task
- Task switching

# Protected mode

## Memory management – segmentation

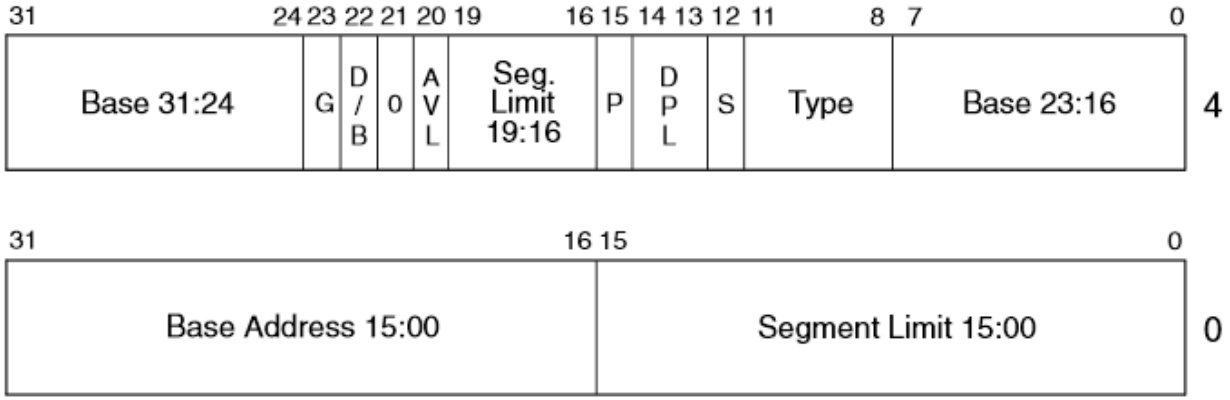
– Segment descriptors

- Are data structures in GDT or LDT that provide the processor with the size a location of a segment, as well as access control and status information.
- Are typically created by compilers, linkers, loaders, or the operating system executive, but not application programs.

# Protected mode

## Memory management – segmentation

– Segment descriptor structure




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

# Protected mode

Memory management – segmentation

- Limit – size of a segment (segments of variable size)
- Base address – starting address of the segment within the linear address space
- S - Application or system descriptor
- Type
  - Growth direction
  - Kind of access

# Protected mode

Memory management – segmentation

- DPL – descriptor privilege level (0 to 3)
- P – the segment is present in main memory
- D/B (segment type specific) – default size for the operations within the segment (32 or 16 bits)
- G (granularity) – determines the scaling of the segment limit field (byte level or 4KB level)

# Protected mode

## Memory management – code and data types

**Table 3-1. Code- and Data-Segment 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-Only, conforming |
| 15 | 1 | 1 | 1 | 1 | Code | Execute/Read-Only, conforming, accessed |

# Protected mode

Memory management – system types

- System segment descriptors
  - Local descriptor-table (LDT) segment descriptor
  - Task-state segment (TSS) descriptor
- Gate descriptors
  - Call-gate descriptor
  - Interrupt-gate descriptor
  - Trap-gate descriptor
  - Task-gate descriptor

# Protected mode

Memory management – system types

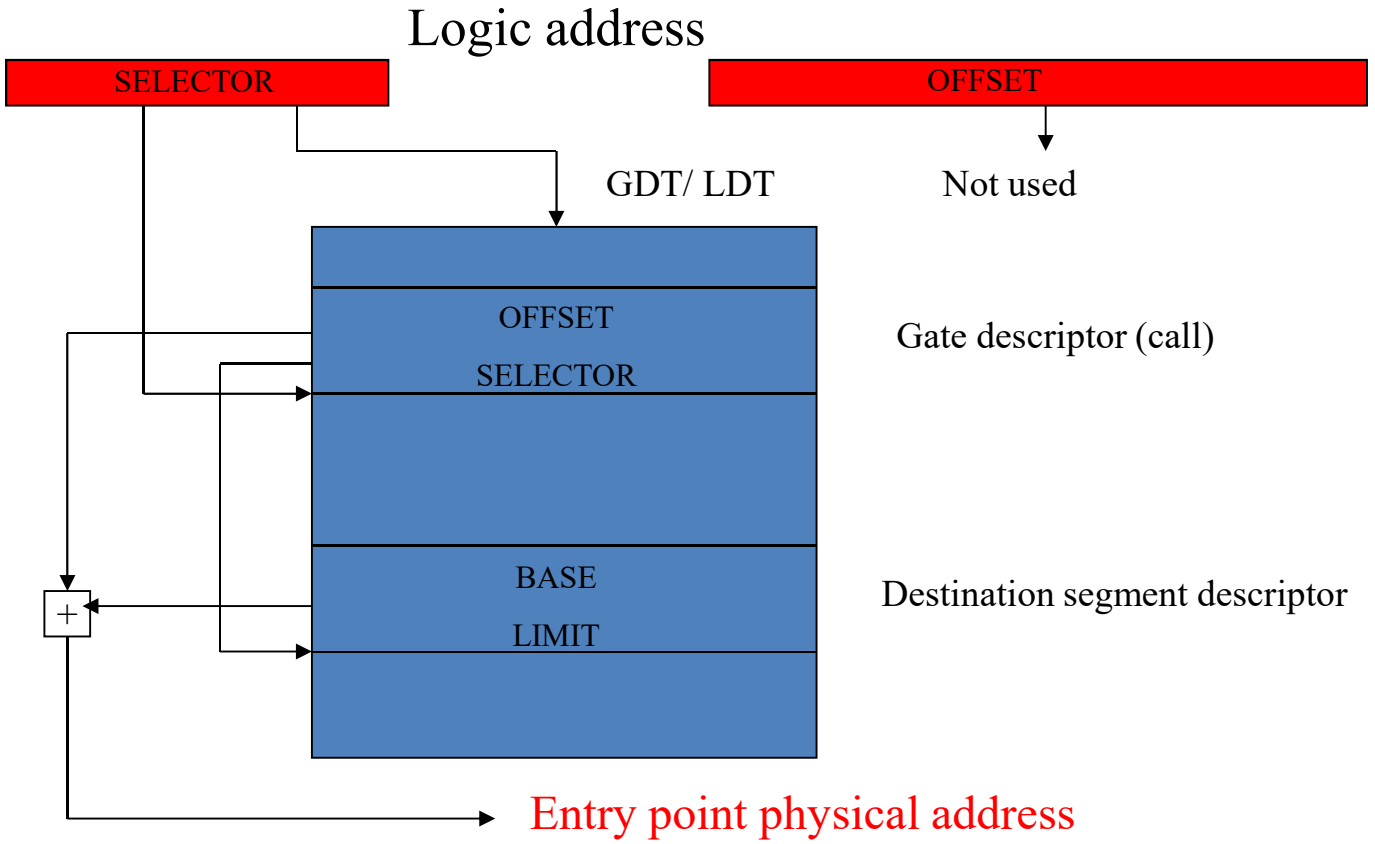
– Gate descriptors

- Pass control flow from one segment to another segment
- Additional control level between source segment and destination segment
- Used by calls, interrupts, traps
- Source segment -> destination segment

# Protected mode

## Memory management – system types

– Gate descriptors



# Protected mode

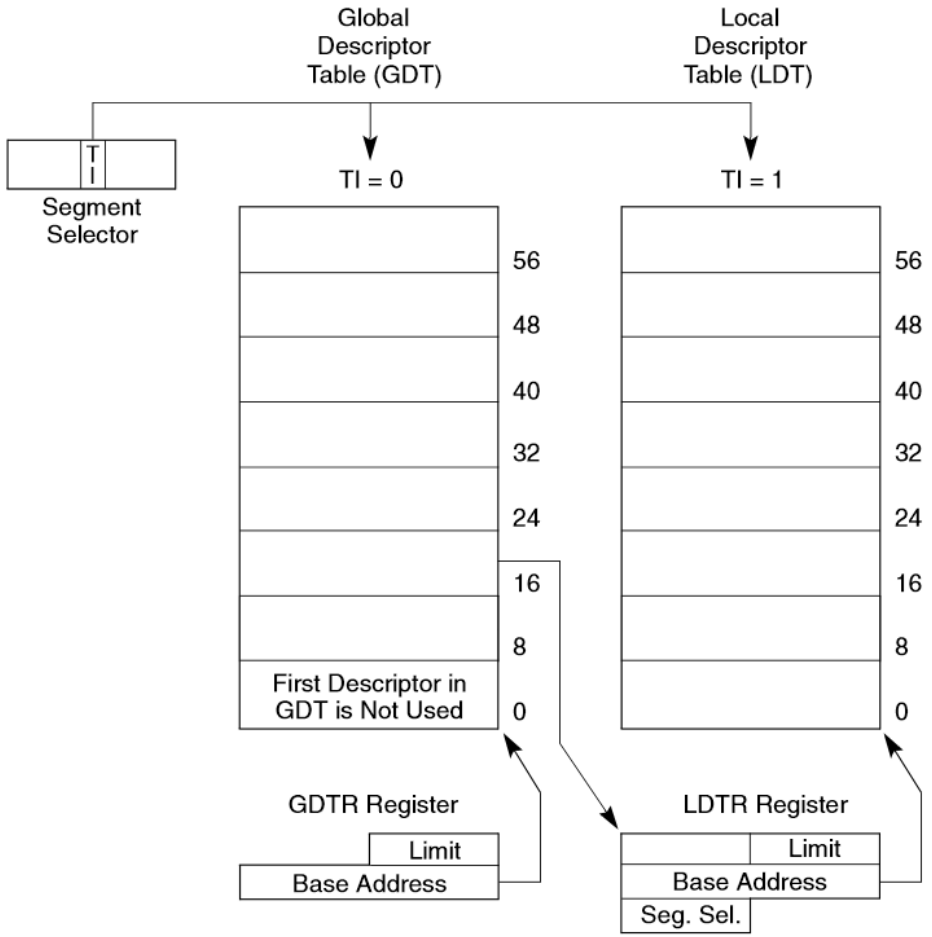
## Memory management – system types

### – TSS – task status segments

- Task switching
  - Transfer the control flow from one task to another
  - Switch task contexts – save the context of the current task and load the context of the next one
  - Each task has allocated one special segment called TSS – It contains the status of the current task
    - » Stack pointers and their descriptors
    - » Content of registers
    - » LDT
    - » I/O map

# Protected mode

## Memory management – segmentation



# Protected mode

## Memory management – segmentation

- Disadvantages
  - Multiple memory accesses for one useful access
  - High memory access latency
- Solution
  - Cache registers holding segment descriptors

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

# Protected mode

Memory management – segmentation

- Advantages
  - Multi-tasking (task isolation)
    - Per task segments allocation
    - Task level protection
  - Kernel resources protection
  - Task errors localization

# Protected mode

Memory management – segmentation

- Every memory access is validated automatically by the processor
  - Segment type / type of access
  - Segment size
  - Segment entry points
  - Privilege levels
    - 0 - highest level (kernel, drivers)
    - 3 – lowest level (user application)

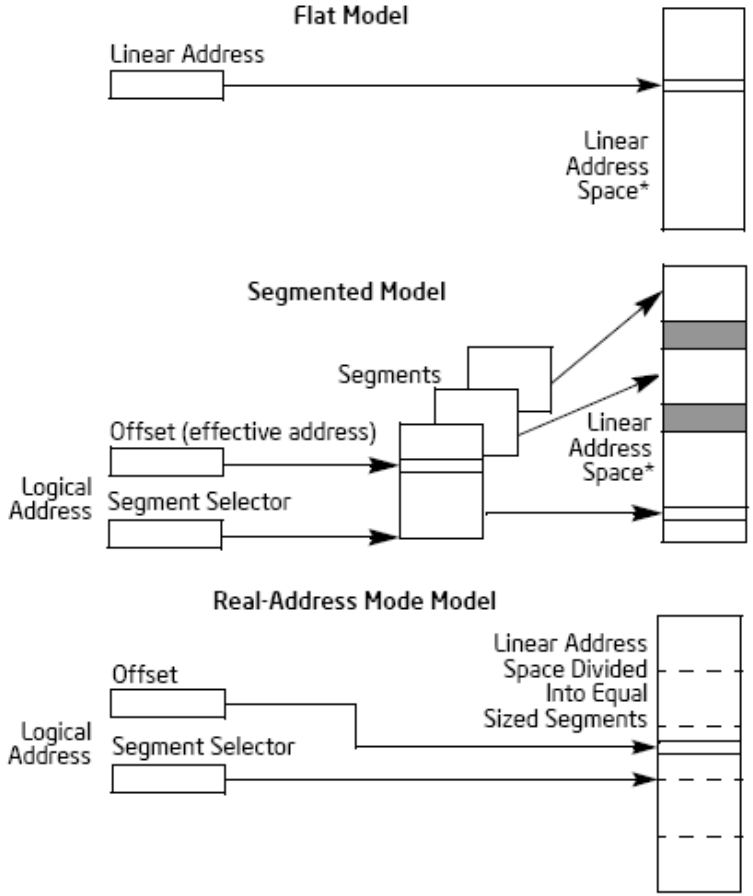
# Protected mode

Memory management – segmentation

- Privilege levels
  - CPL – current privilege level of the running program
  - DPL – descriptor privilege level of the accessed segment
  - RPL – requested privilege level of data segment
- Data access
  - Task privilege level = max(CPL, RPL)
  - Access max(CPL, RPL) <= DPL
  - Exceptions

# Protected mode

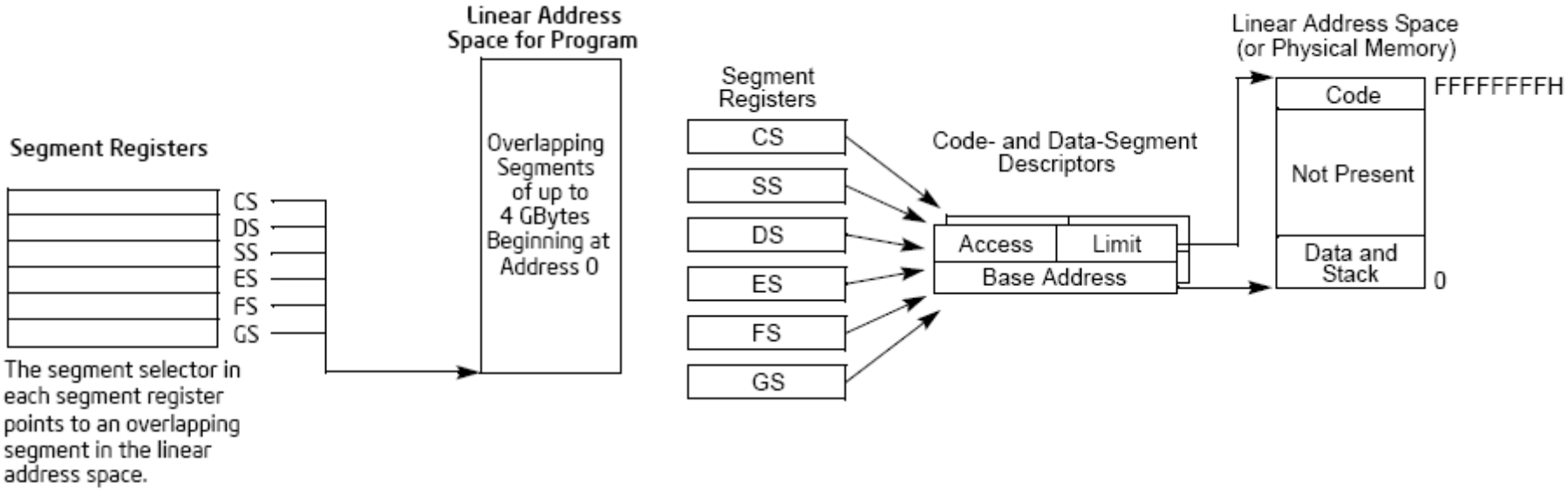
## Memory management – segmentation



# Protected mode

## Memory management – segmentation

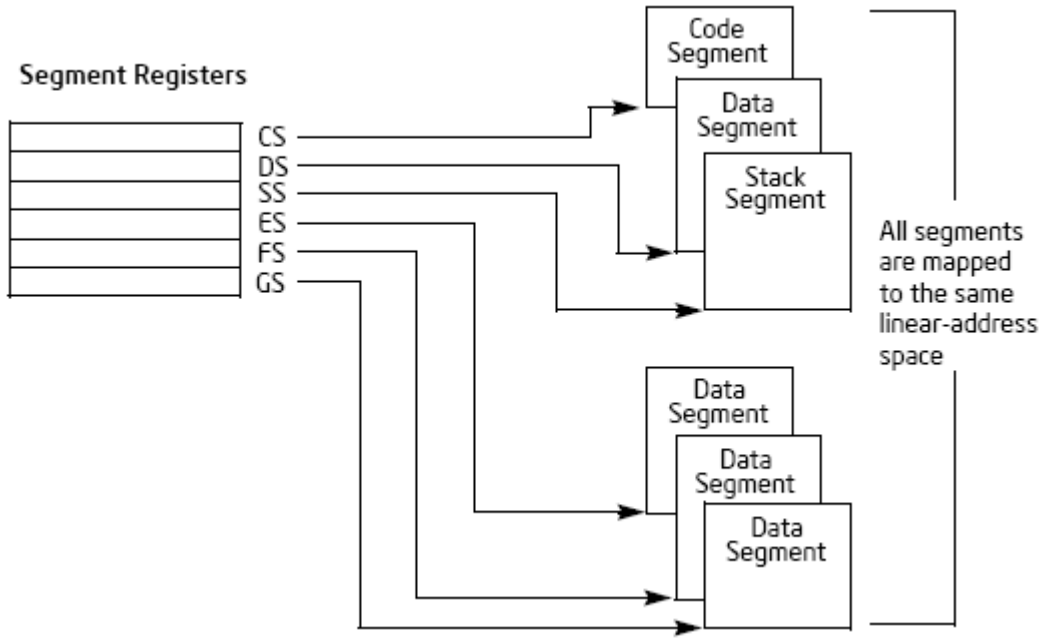
– Flat model



# Protected mode

Memory management – segmentation

– Segmented model



# Protected mode

Memory management – paging

- Implementation of virtual memory concept
- Further translates the linear addresses into physical addresses using pagination
- Page sizes much more lower than segment sizes
  - 4KB pages
- Hierarchical paging structure

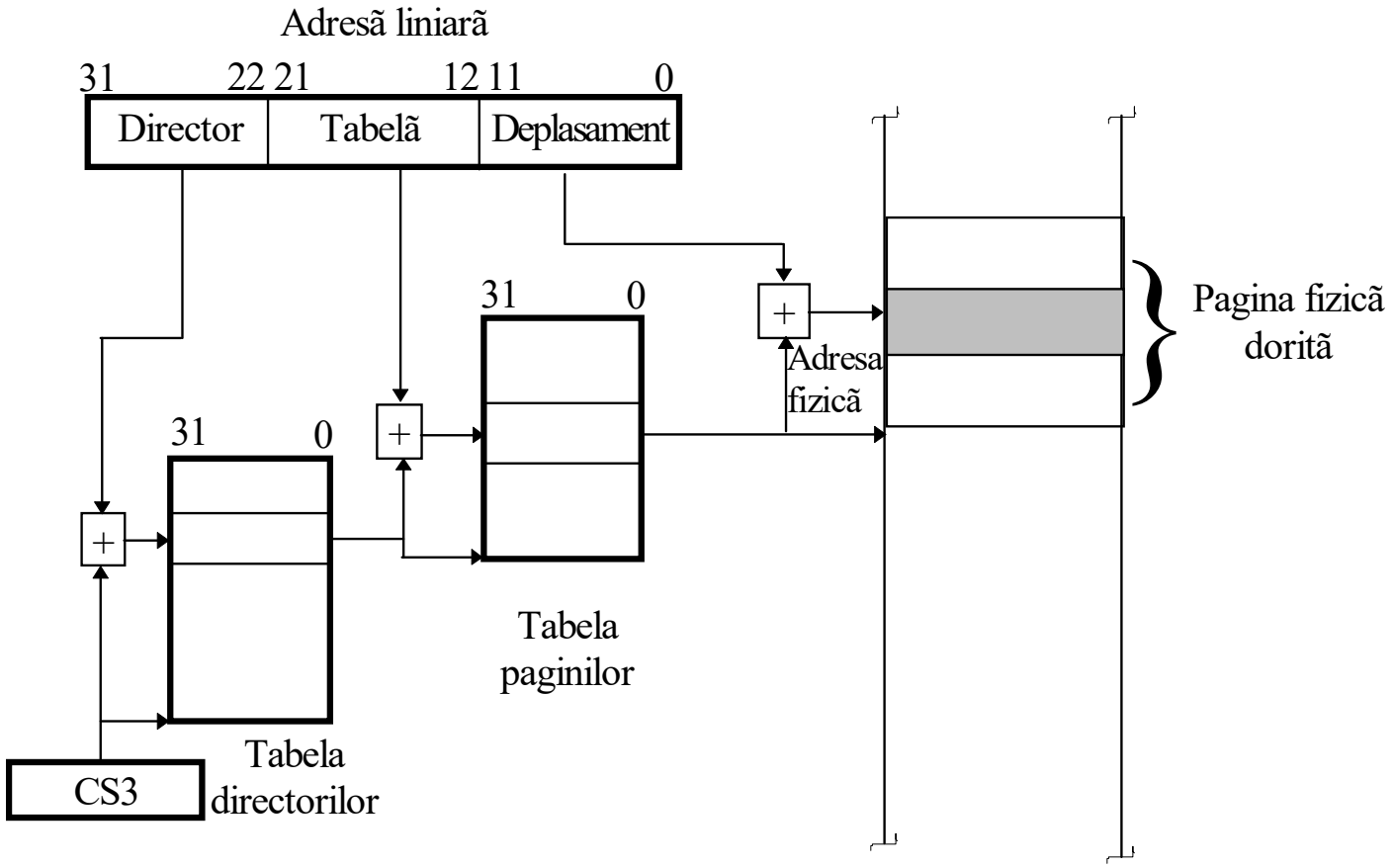
# Protected mode

Memory management – paging

– Every paging structure is 4KB and contains 1024 entries describing a memory page or other paging structures

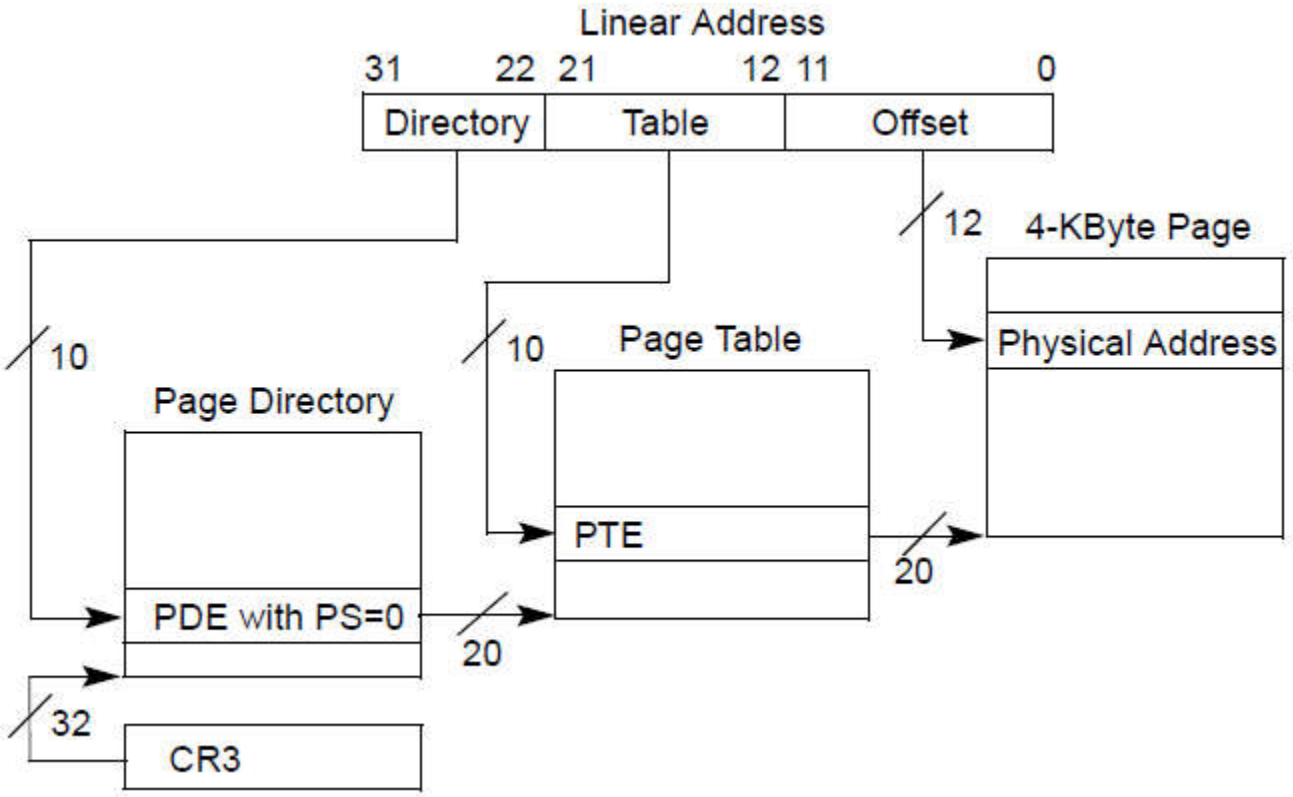
# Protected mode

## Memory management – paging



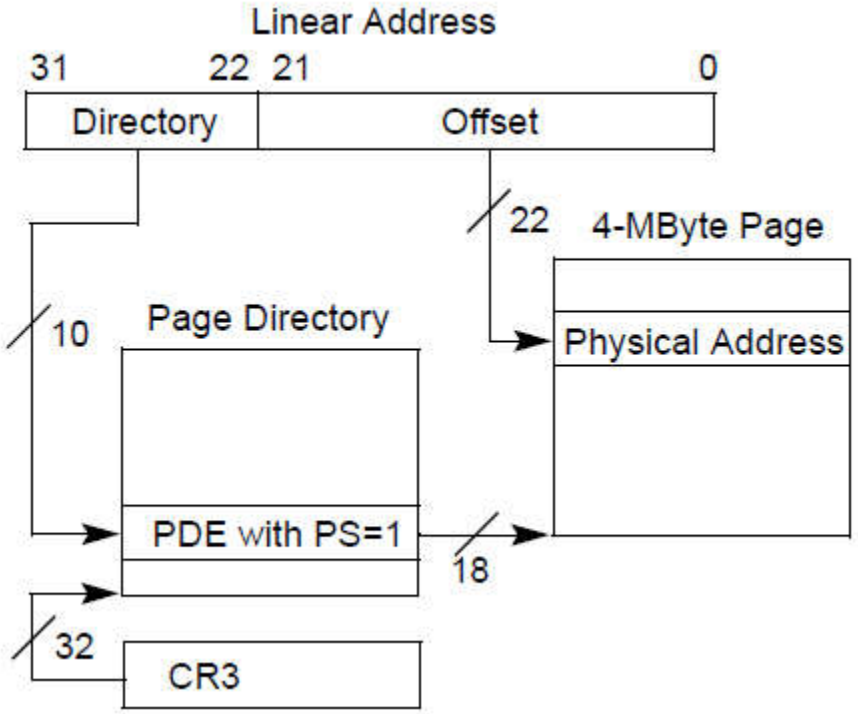
# Protected mode

## Memory management – paging



# Protected mode

Memory management – paging



# Protected mode

Memory management – paging

Advantages:

- 4 Ko 1024 x 1024 = 1 M pages can be managed;
- Paging tables are 4 Ko pages too;
- Small compact memory blocks to be managed:
  - 4 Ko faţă de  un mecanism pe un singur nivel, cu pagini de 4 Ko, pentru care ar fi fost necesară o tabelă cu 1 M int şi considerând 4 octeţi/ intrare ar fi fost necesară o zonă de memorie compactă de 4 Mo.

Disadvantages:

- Additional memory accesses – two new memory reads for a data transfer
  - Increased delay;
- Solution: cache memory for translated addresses:
  - TLB – translation lookaside buffer;
  - Translated addresses of most recent pages;

# Protected mode

```
g_desc      struc           ; segment descriptor
mit         dw      0       ; segment length
se_lo               dw      0       ; low word
se_hi               db      0       ; high word
cess        db      0       ; access rights
            dw      0       ; reserved
g_desc      ends

t_desc              struc   ; interrupt descriptor
set         dw      0       ; offset of interrupt handler
lect        dw      0       ; segment selector in GDT or LDT
            db      0       ; unused
ags         db      0       ; flags
            dw      0       ; reserved
t_desc              ends
```

# Protected mode

```
gdt0   seg_desc<GDT_LEN,,,92h,>   ; GDT null descriptor
gdt1   seg_desc<CS_LEN,,,9ah,>    ; code descriptor
gdt2   seg_desc<DS_LEN,,,92h,>    ; data descriptor
gdt3   seg_desc<SS_LEN,,,97h,>    ; stack descriptor
gdt4   seg_desc<0ffffh,,,92h,>    ; video memory descr.
gdt5   seg_desc<0ffffh,,,92h,>    ; user descriptor
GDT_LEN       equ    $-gdt0

idt_start     int_desc<IDT_LEN,,,,>      ; IDT
idt0   int_desc      IDT_LEN dup (<offset int_proc,8,,86h,>)
```

# Protected mode

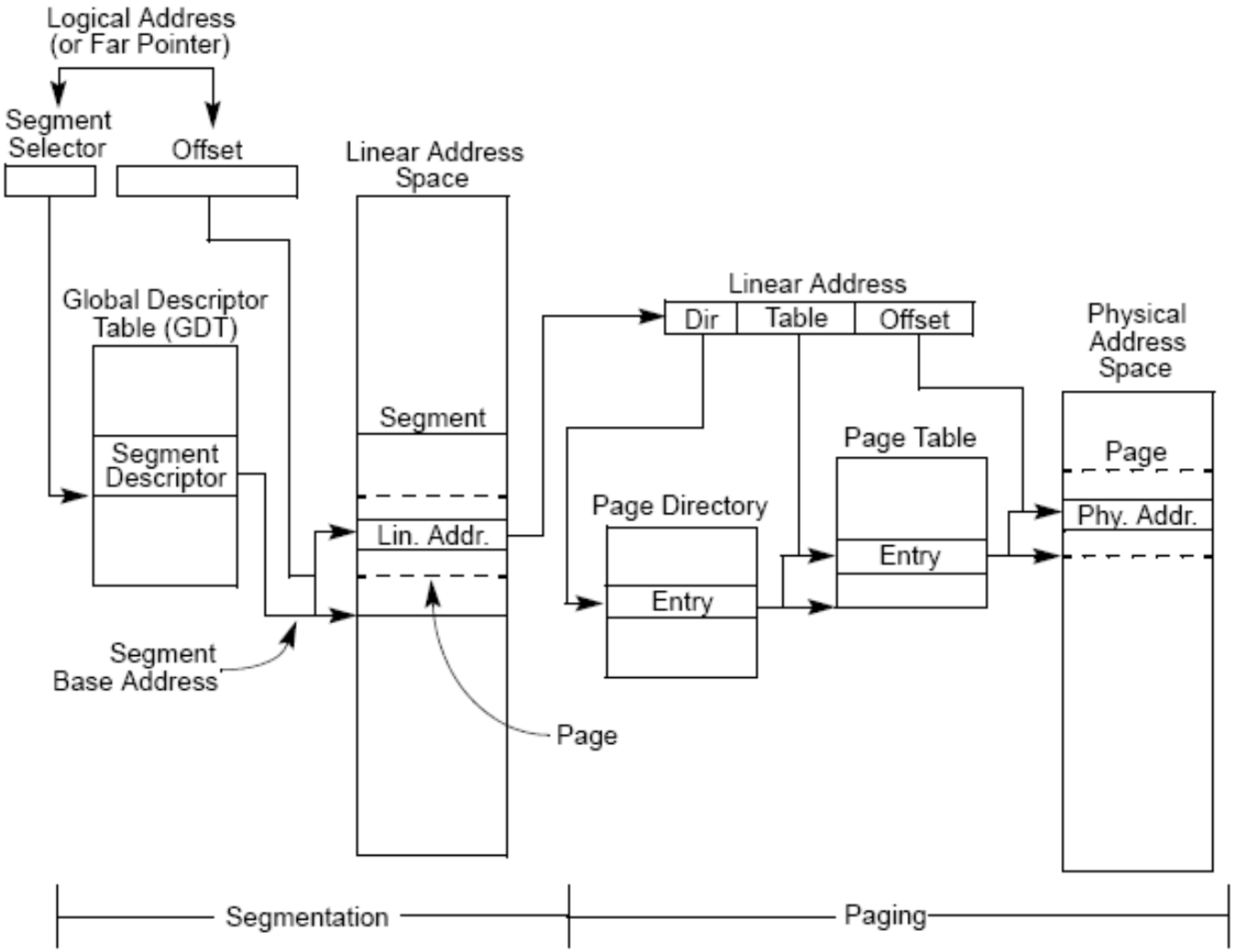
```
        mov     dx,ds
        mov     cx,offset gdt0
        call    calc24
        mov     gdt0.base_lo,dx
        mov     gdt0.base_hi,cl

        mov     dx,cs
        xor     cx,cx
        call    calc24
        mov     gdt1.base_lo,dx
        mov     gdt1.base_hi,cl

        lgdt    gdt0
        mov     ax,1
        lmsw    ax                  ; enter protected mode

        jmpfar  continue_2,8
tinue_2:mov     ax,3*8
        mov     ss,ax
        mov     sp,offset top_off
```

# Protected mode



# Protected mode