Lecture 2

Memory management in PM

Memory management of the IA-32 architecture provides:
- segmentation,
- paging.

Segmentation – mechanism of isolating code, data and stack modules of multiple programs enabling their operation without mutual interference.

Paging – mechanism for implementing virtual-memory systems where sections of programs are mapped into physical memory when needed. Use of paging is optional. It can be switched on/off with 31 bit in CR0 control register (see Fig. 2.2).

```
mov eax, cr0
or al, 01h
mov cr0, eax
```

Turning PM on

```
mov eax, cr0
and al, 0feh
mov cr0, eax
```

Turning PM off

Fig. 2.1 Sample codes turning paging on and off

```
mov eax, cr0
or eax, 80000000h
mov cr0, eax
```

Turning paging on

```
mov eax, cr0
and eax, 7fffffffh
mov cr0, eax
```

Turning paging off

Fig. 2.2 Sample codes turning the PM on and off

With the aid of CR0 register (bit 0) we are able to switch the PM on and off (see Fig. 2.1).

Memory management in PM

Segmentation

Segmentation provides mechanisms for dividing the processor's addressable memory space (here called the **linear address space**) into protected address spaces called **segments**. Segments can hold: program code, data and stack, and system data structures.

Memory segments can be assigned: attributes of **privilege**, **base address**, **limit** and **usage**. That information defining memory segment is stored in **segment descriptor**.

Segment descriptors are stored in **descriptor tables**: **global** (GDT) and **local** (LDT) and are identified with **segment selectors**.

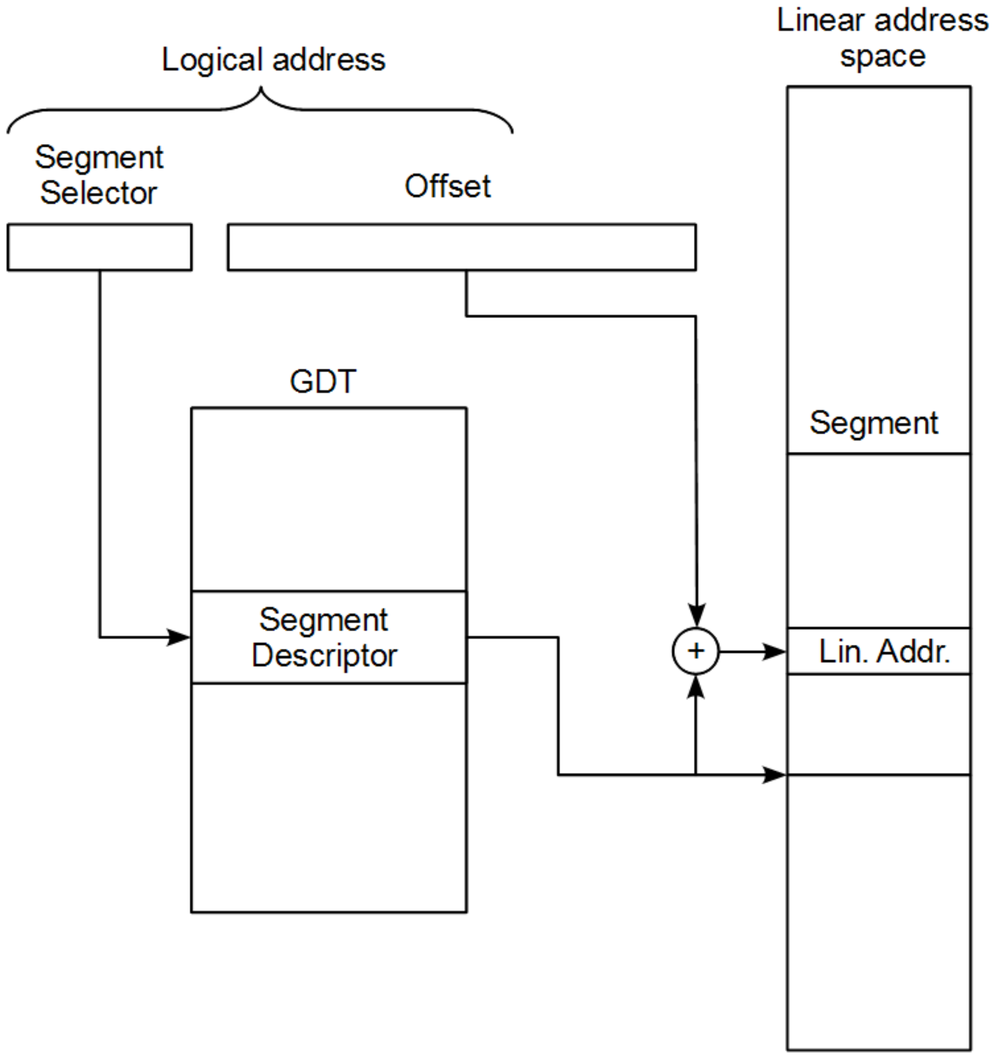


Fig. 2.3 Segmentation

Memory management in PM

Segment descriptor format

Segment descriptor is an 8 bytes long structure that contains one memory segment's: base address (32-bits), limit (20-bits) and access rights (see Fig. 2.4 and 2.5).

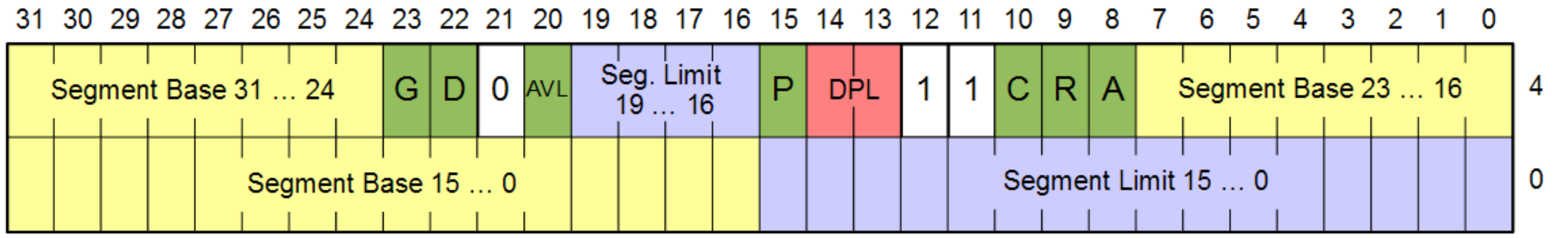


Fig. 2.4 32-bit Code Segment Descriptor

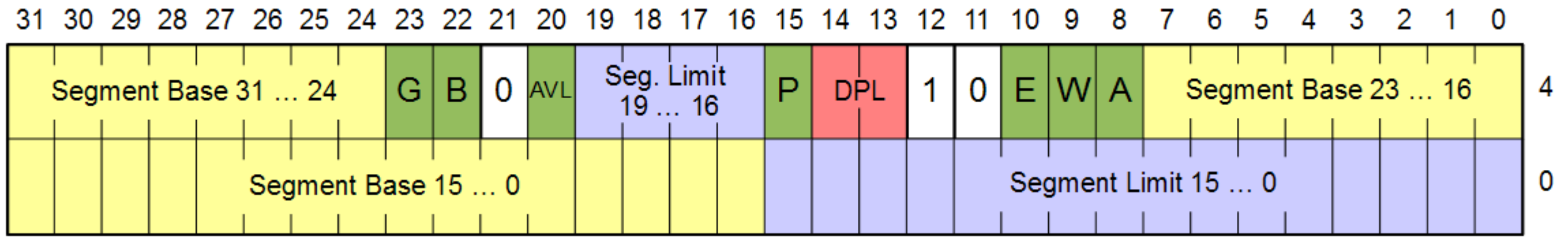


Fig. 2.5 32-bit Data/Stack Segment Descriptor

Definition of descriptor fields:

1. **Segment base address** – 32-bit value defining the base address of the segment in the linear address space.

Memory management in PM

2. **Segment limit** – 20-bit field defining the largest offset of the segment. The limit can be byte (G=0) or 4kB page (G=1) granular.

3. **G** (Granularity) – indicates whether the limit is byte (G=0) or 4kB page (G=1) granular. With byte granularity offset can be adjusted with byte resolution. Hence the maximum segment size is 1MB. For page-granularity the unit change of offset value shifts the linear address by 4kB and the maximum segment size is 4GB.

4. **D** (Default) – indicates whether the segment is 32-bit (D=1) or 16-bit (D=0).

5. **E** (Expansion Direction) – used with data segments, indicates whether the segment extends from its base address up to base address+limit (E=0), or from the maximum offset down to the limit (E=1). A data stack is usually the *expand-down* (E=1) segment what enables the dynamic change of its size.

6. **B** (Big) – for data segments, indicates the maximum offset as 0ffffffffh (B=1) and 0000ffffh (B=0). This bit is significant only for *expand-down* segments.

7. Access rights:

   a. **P** (Present) – segment is in physical memory (P=1) or not (P=0). Used by virtual memory managers.

   b. **A** (Accessed) – indicates whether the segment was accessed (A=1) since the last time bit A was cleared. In virtual memory systems operating upon segments this bit can determine segment usage.

Memory management in PM

c. **DPL** (Descriptor Privilege Level) – indicates privilege level of the segment as a number 0, 1, 2 or 3.

DPL Field	Description
00	Level 0 most privileged (supervisor, kernel)
01	Level 1
10	Level 2
11	Level 3 least privileged (user application)

Tab. 2.1 Privilege levels

The DPL of current code segment indicates Current Privilege Level (**CPL**).

d. **R** (Readable) – in case of code segments indicates whether the code segment is readable (R=1) or not (R=0). Code segments are always executable.

e. **C** (Conforming) – for code segments only, indicates wether the CPL changes when the segment is called from lesser privilege level (C=0) or not (C=1).

f. **W** (Writable) – for data segments indicates whether the segment is writable (W=1) or not (W=0). Data segments are of course always readable. Data segments for stack must be writable.

g. **AVL** – available for use with system software.

Memory management in PM

Examples of segment descriptors

Code segment

Real mode segment address: 4b10h:0000h
Segment size: 1000h B
Access rights: readable, non-conforming, present, granularity 1 B, 32-bits, supervisor level

Linear (32-bit) base address: 16*4b10h+0000h=0004b100h
Limit: 1000h – 1=0fffh

Descriptor: **dw 0fffh, b100h, 9a04h, 0040h**

Data segment

Real mode segment address: 5cf0h:0000h
Segment size: 4000h B
Access rights: writable, present, granularity 1 B, 32-bits, user level, expand up

Linear (32-bit) base address: 16*5cf0h+0000h=0005cf00h
Limit: 4000h – 1=3fffh

Descriptor: **dw 3fffh, 0cf00h, 0f205h, 0040h**

Memory management in PM

Global Descriptor Table

The Global Descriptor Table holds an array of segment descriptors. Its address and limit are located in **GDTR** register which can be written with **LGDT** instruction. There is only one GDT.

The GDT can hold up to 8192 descriptors including obligatory NULL descriptor.

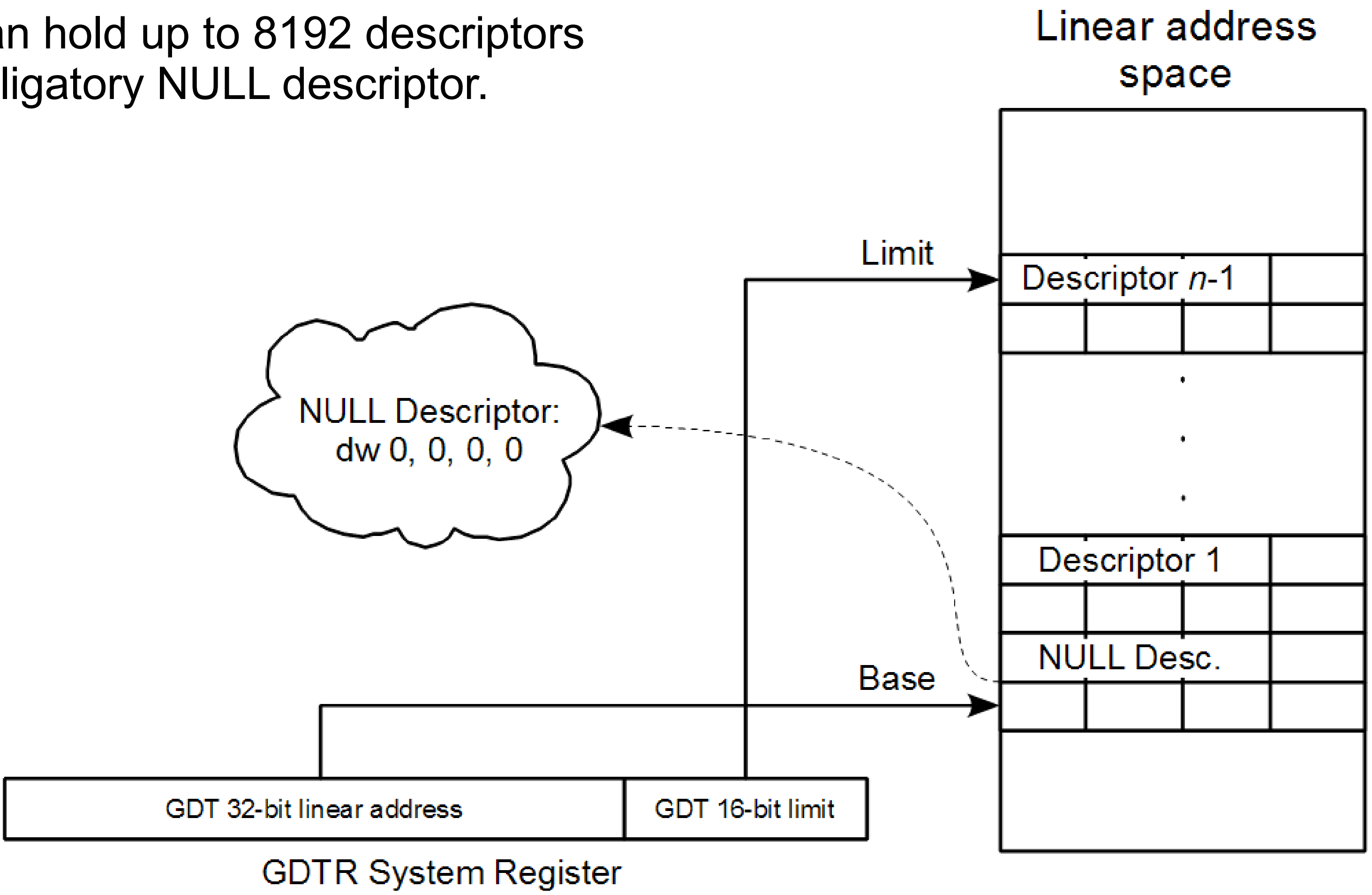


Fig. 2.6 Global Descriptor Table

Memory management in PM

Selector

Descriptors are identified by 16-bit selectors of the following form.

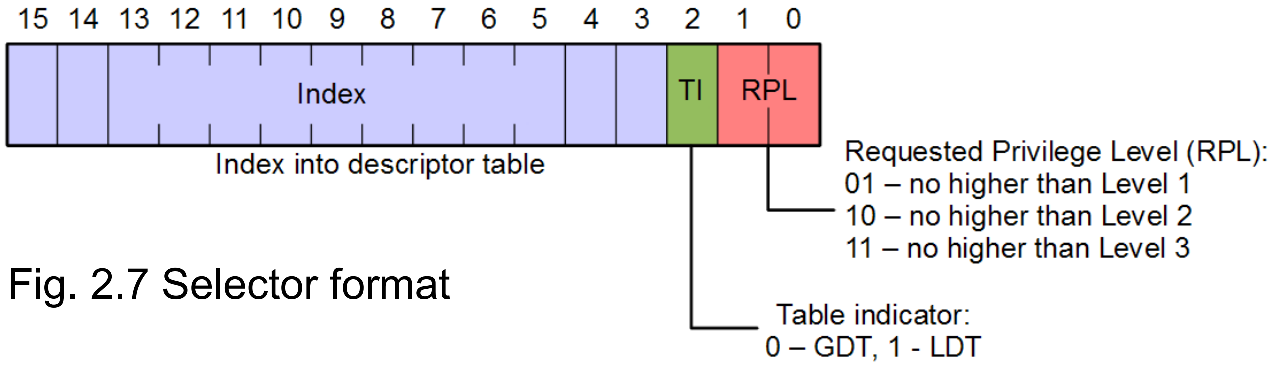

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Index
TI
RPL
Index into descriptor table
Requested Privilege Level (RPL):
01 – no higher than Level 1
10 – no higher than Level 2
11 – no higher than Level 3
Table indicator:
0 – GDT, 1 - LDT

Fig. 2.7 Selector format

Local descriptor table

Each application (task) can have its own address space defined by descriptors written in its Local Descriptor Table (**LDT**).

Current LDT is indicated (and defined by descriptor in GDT) by its selector loaded into **LDTR** System Register with the aid of **LLDT** instruction.

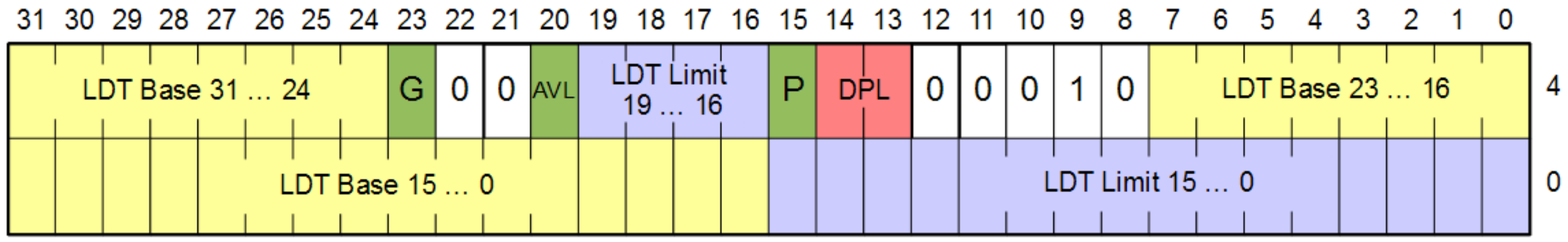
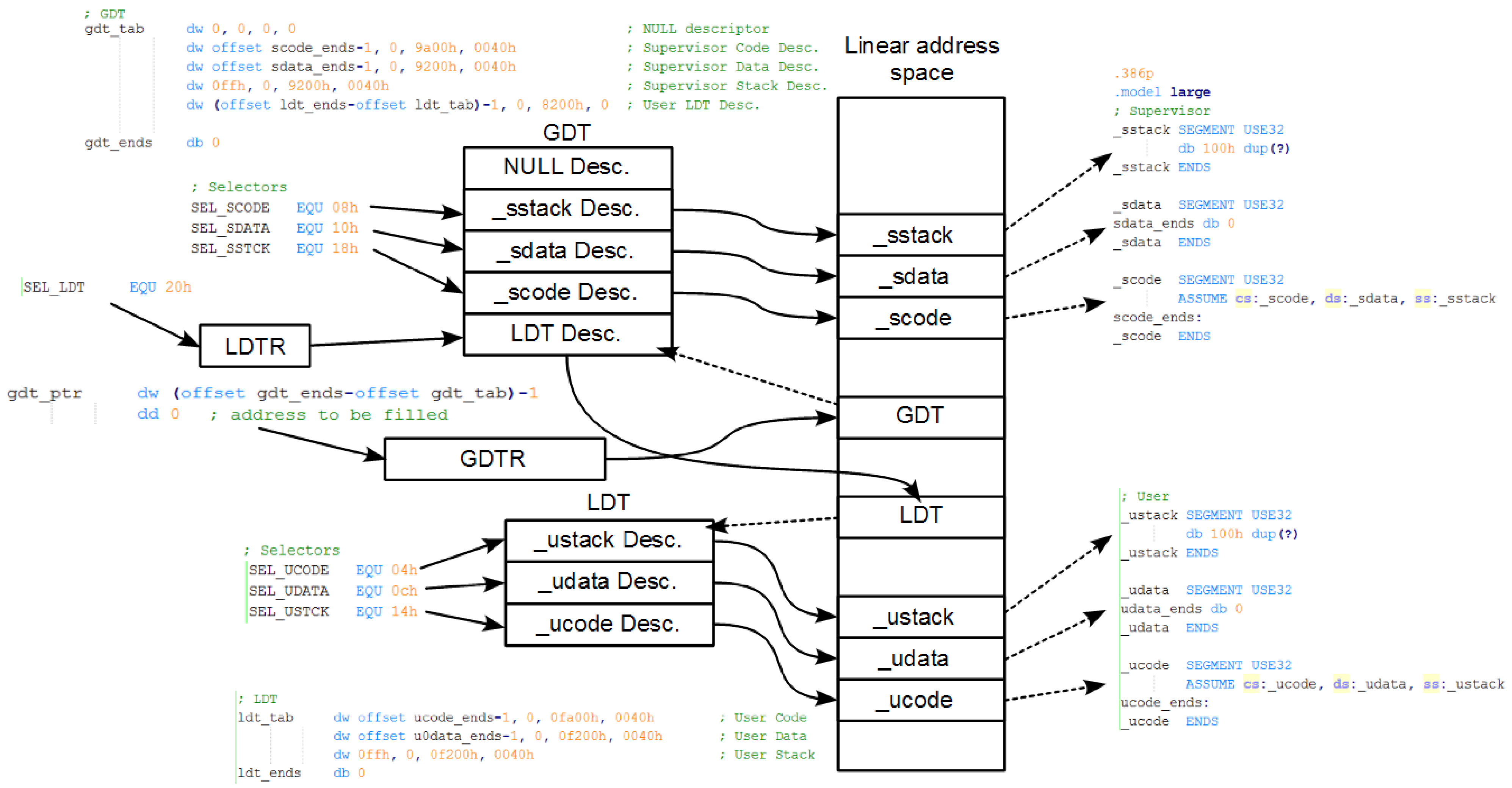


Fig. 2.8 LDT Descriptor

Memory management in PM

Exemplary application - descriptors structure



```
; GDT
gdt_tab     dw 0, 0, 0, 0                                ; NULL descriptor
            dw offset scode_ends-1, 0, 9a00h, 0040h      ; Supervisor Code Desc.
            dw offset sdata_ends-1, 0, 9200h, 0040h      ; Supervisor Data Desc.
            dw 0ffh, 0, 9200h, 0040h                     ; Supervisor Stack Desc.
            dw (offset ldt_ends-offset ldt_tab)-1, 0, 8200h, 0  ; User LDT Desc.

gdt_ends    db 0
```

```
; Selectors
SEL_SCODE   EQU 08h
SEL_SDATA   EQU 10h
SEL_SSTCK   EQU 18h
```

```
SEL_LDT     EQU 20h
```

```
gdt_ptr     dw (offset gdt_ends-offset gdt_tab)-1
            dd 0   ; address to be filled
```

```
; Selectors
SEL_UCODE   EQU 04h
SEL_UDATA   EQU 0ch
SEL_USTCK   EQU 14h
```

```
; LDT
ldt_tab     dw offset ucode_ends-1, 0, 0fa00h, 0040h     ; User Code
            dw offset u0data_ends-1, 0, 0f200h, 0040h    ; User Data
            dw 0ffh, 0, 0f200h, 0040h                    ; User Stack
ldt_ends    db 0
```

```
.386p
.model large
; Supervisor
_sstack SEGMENT USE32
        db 100h dup(?)
_sstack ENDS

_sdata  SEGMENT USE32
sdata_ends db 0
_sdata  ENDS

_scode  SEGMENT USE32
        ASSUME cs:_scode, ds:_sdata, ss:_sstack
scode_ends:
_scode  ENDS
```

```
; User
_ustack SEGMENT USE32
        db 100h dup(?)
_ustack ENDS

_udata  SEGMENT USE32
udata_ends db 0
_udata  ENDS

_ucode  SEGMENT USE32
        ASSUME cs:_ucode, ds:_udata, ss:_ustack
ucode_ends:
_ucode  ENDS
```

Fig. 2.9 Exemplary descriptor structure for supervisor/user application

Memory management in PM

Privilege Levels Indicators

CPL (Current Privilege Level) – privilege level of currently executing program or task,

DPL (Descriptor Privilege Level) – privilege level of segment. The DPL is interpreted depending on the type of segment, e. g.:

- Data segment: it is numerically highest privilege that a program or task can have to be allowed to access the segment.

- Code segment (conforming): indicates the numerically lowest privilege level that a program or task can have to be allowed to access the segment.

RPL (Requested Privilege Level) – it is an override privilege level that is assigned to segment selectors. The processor checks RPL and CPL to determine if access is allowed. That is, if RPL is numerically greater than CPL it overrides the CPL, and vice versa. The RPL can be used to ensure that privileged code does not access segments with use of selectors passed, for example as procedure arguments, from lower privileged code unless the lower privilege code is not allowed to (for further information refer to *Intel Manuals*).

Memory management in PM

Access rights checks during memory referencing

Memory read/write, execute
- destination must be located in readable/writable segment,
- destination address must be within the limits of referred to segment,
- addressing with NULL descriptor is not allowed,
- in case of execution transfer the destination segment must be of executable type (transfer between different privilege levels will be discussed during the forthcoming lectures),
- during read/write operation RPL and CPL <= DPL.

Violation of any of above rules results in rise of exception.

Loading segment registers with selectors
- segment selector index must be within limits of descriptor table,
- in case of SS: segment selector cannot be NULL, RPL=CPL, DPL=CPL, segment must be present and must be writable data segment,
- in case of DS, ES, FS, GS: segment must be readable data segment, RPL, CPL <= DPL, segment must be present.

Violation of any of above rules results in rise of exception.

Requirements for data, stack and code segments in real mode

When switching back to the real mode it is required to provide the selectors for: 16-bit, readable, present, 1B granularity, maximum limit 0ffffh, executable privilege 0 code segment in CS, and 16-bit, present, writable, 1B granularity, 0ffffh limit, expand up, privilege 0 data segment in DS, ES, FS, GS and SS registers.

Memory management in PM

Memory paging

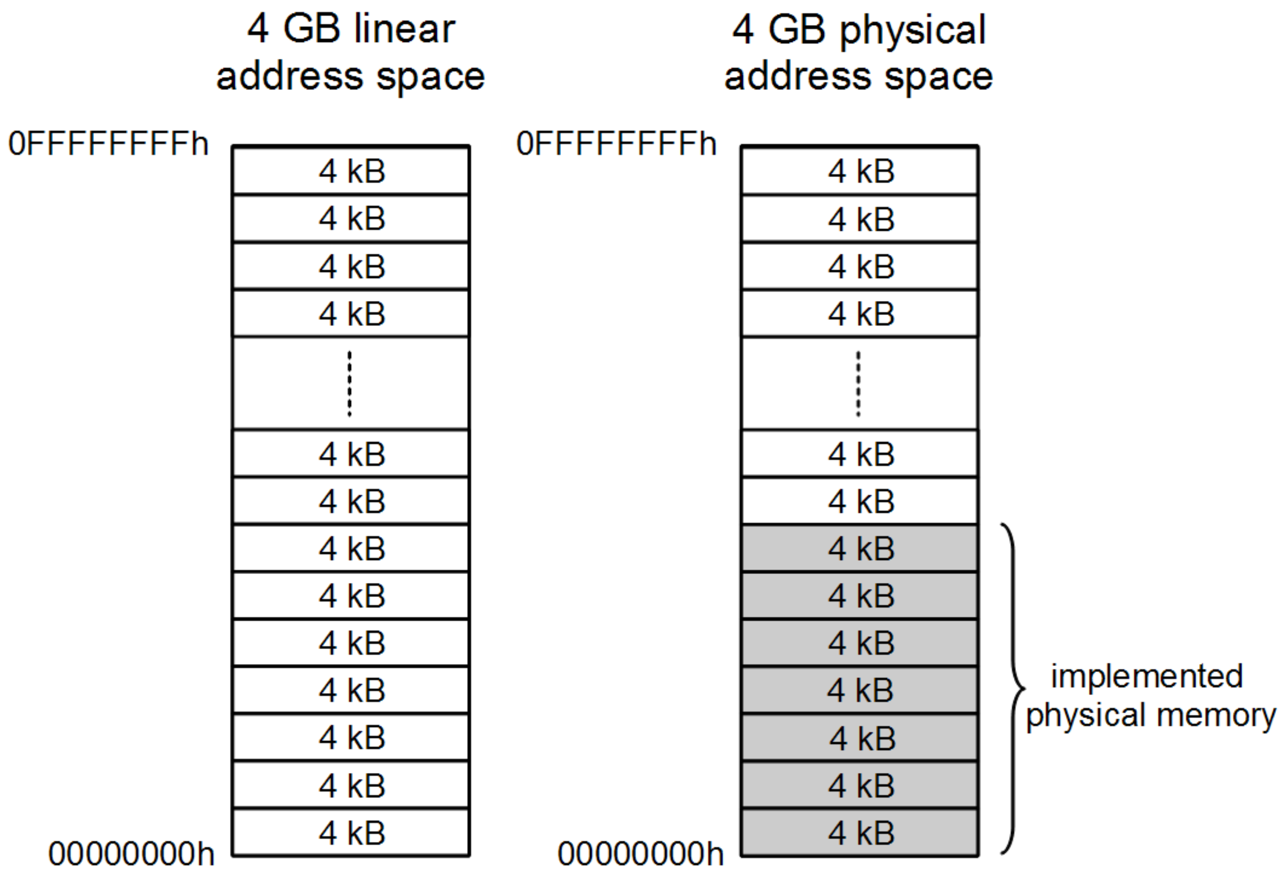
The 4 GB linear address space is divided into 1048576 pages of 4 kB each (see Fig. 2.10).

Usually virtual memory is much larger than implemented one.

When accessing memory page that is not in physical memory an exception is generated. The exception allows the supervisor software to load required page (e.g. from HDD) transparently to the user program.

The "new" page is loaded into physical memory in place of some other page which must be written on HDD.

During such operation address translation is required.



4 GB linear address space
4 GB physical address space
0FFFFFFFFh
0FFFFFFFFh
4 kB
00000000h
00000000h
implemented physical memory

Fig. 2.10 Paged virtual and physical memory

Memory management in PM

Address translation process

The address translation process replaces the upper 20 bits of linear address (**virtual page address**) with 20-bit value of the **physical page address**. The translated value is obtained from hierarchical translation tables (see Fig. 2.11).

The page directory includes 1024 entries defining the location of page tables

Each of page tables entries indicates the location of one 4 kB long memory page.

The CR3 register contains 20-bit address of page directory.

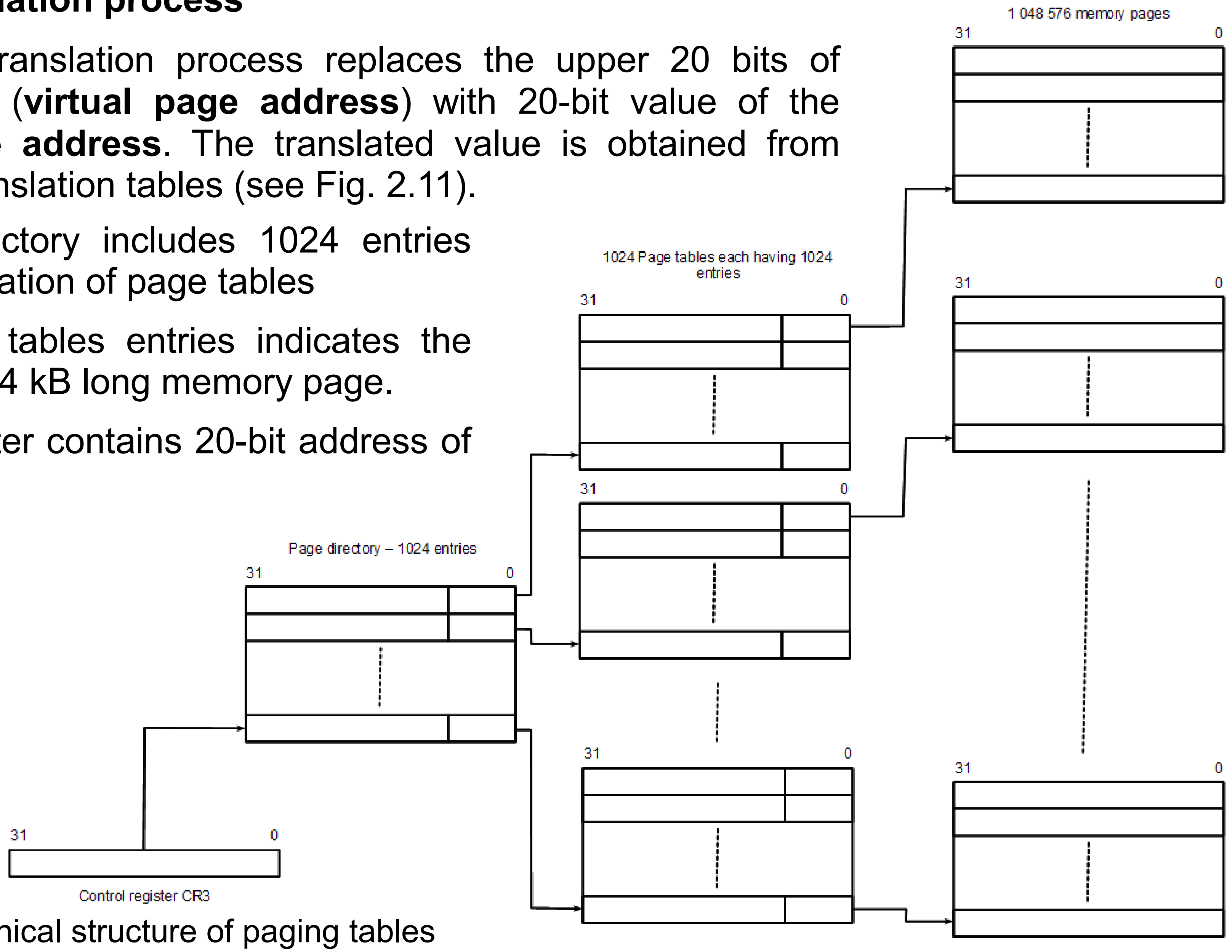

1 048 576 memory pages
1024 Page tables each having 1024 entries
Page directory – 1024 entries
Control register CR3
31
0

Fig. 2.11 Hierarchical structure of paging tables

Memory management in PM

Page directory entry

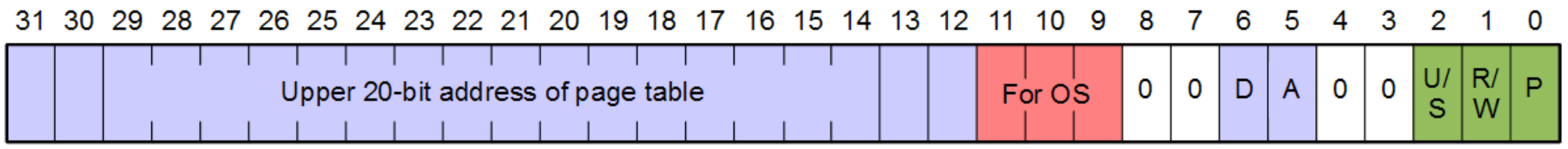


Fig. 2.12 Structure of page directory entry

1. **Page table address** – upper 20 bits of page table address. Lower 12 bits are all 0.

2. **For OS** – for use of operating system.

3. **D** (Dirty) – indicates whether any of 1024 page table pages was written (D=1) or not (D=0). If D=1 then the supervisor program must write on disk one (several) page(s) back before replacing the page(s) in memory.

4. **A** (Accessed) – indicates whether any of 4 MB pages has been written/read. Useful for statistics of page usage.

5. **U/S** (User/Supervisor) – enables protection for 1024 pages described by the page table.

6. **R/W** (Read/Write) – indicates write protection for 1024 pages described by the page table.

7. **P** (Present) – indicates whether the page table is present in physical memory (P=1) or not (P=0).

Memory management in PM

Page table entry

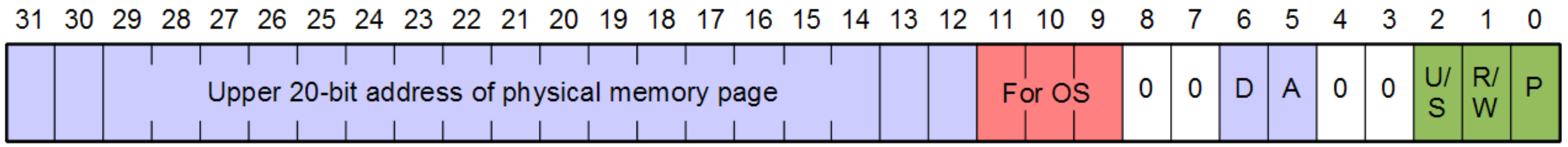


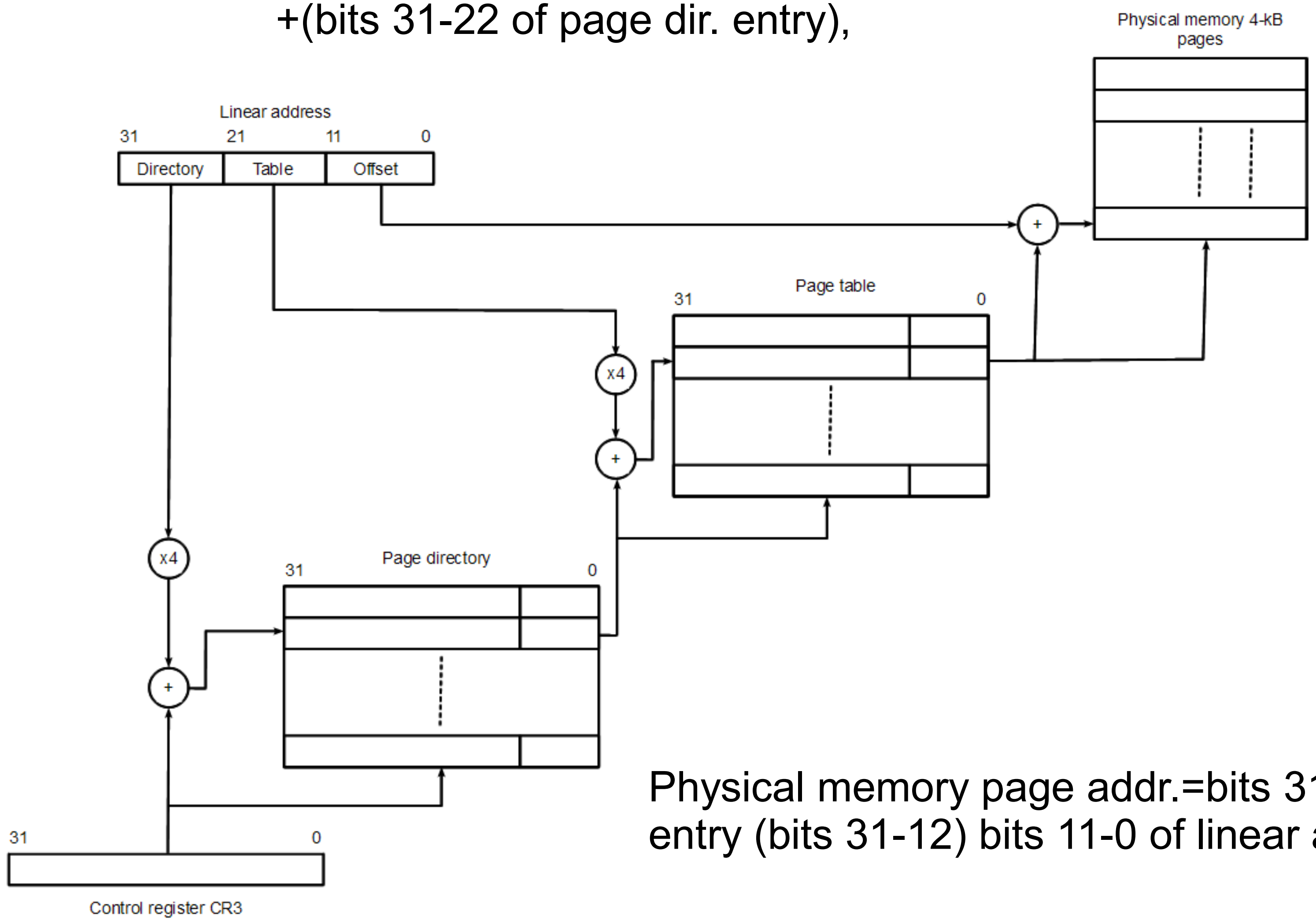
Fig. 2.13 Structure of page table entry

1. **Page address** – upper 20 bits of page address. Lower 12 bits are all 0.
2. **For OS** – for use of operating system.
3. **D** (Dirty) – indicates whether any of 1024 pages was written (D=1) or not (D=0). If D=1 then the supervisor program must write on disk the page back before replacing it in memory.
4. **A** (Accessed) – indicates whether the page has been written/read. Useful for statistics of page usage.
5. **U/S** (User/Supervisor) – enables protection for physical memory page described by the page table entry.
6. **R/W** (Read/Write) – indicates write protection for the page.
7. **P** (Present) – indicates whether the page is present in physical memory (P=1) or not (P=0).

Memory management in PM

Address translation process (cont.)

Page dir. entry addr. = bits 31-12 from CR3 (bits 31-12) bits 11-2 from linear addr. (bits 31-22),
Page table entry addr. = bits 31-12 from dir. entry (bits 31-12) bits 11-2 of linear addr. +(bits 31-22 of page dir. entry),



Physical memory page addr.=bits 31-12 from table entry (bits 31-12) bits 11-0 of linear addr..

Fig. 2.14 Translating linear address to the physical one

Memory management in PM

Address translation (cont.)

In order to speed up the translation process (i.e. to minimize the number of references to memory that holds translation tables) **Translation Lookaside Buffer** (TLB) is used. Such buffer is designed as four-way associative memory and holds addresses and privileges of thirty two physical memory pages (see Fig. 2.15).

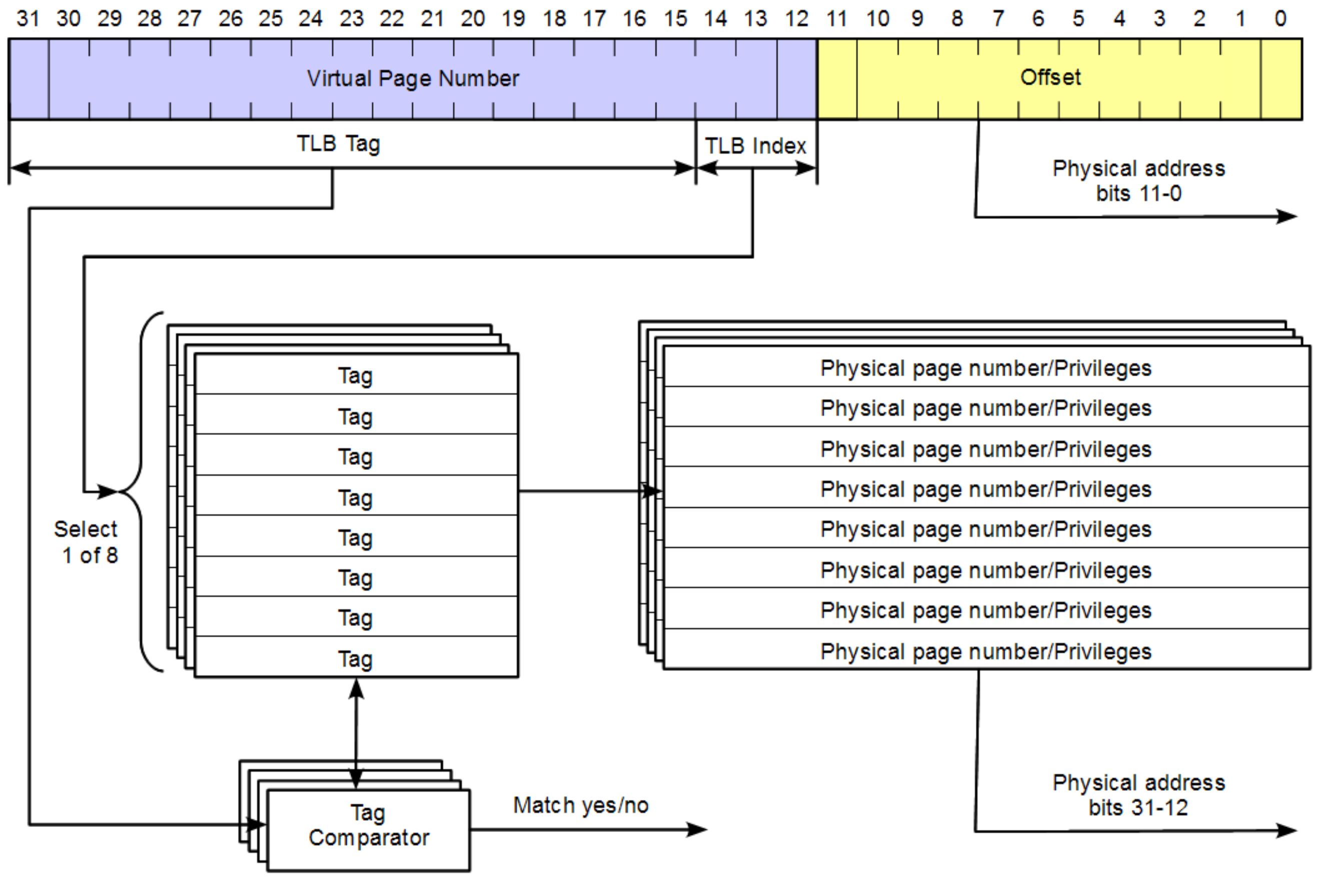


Fig. 2.15 TLB organization