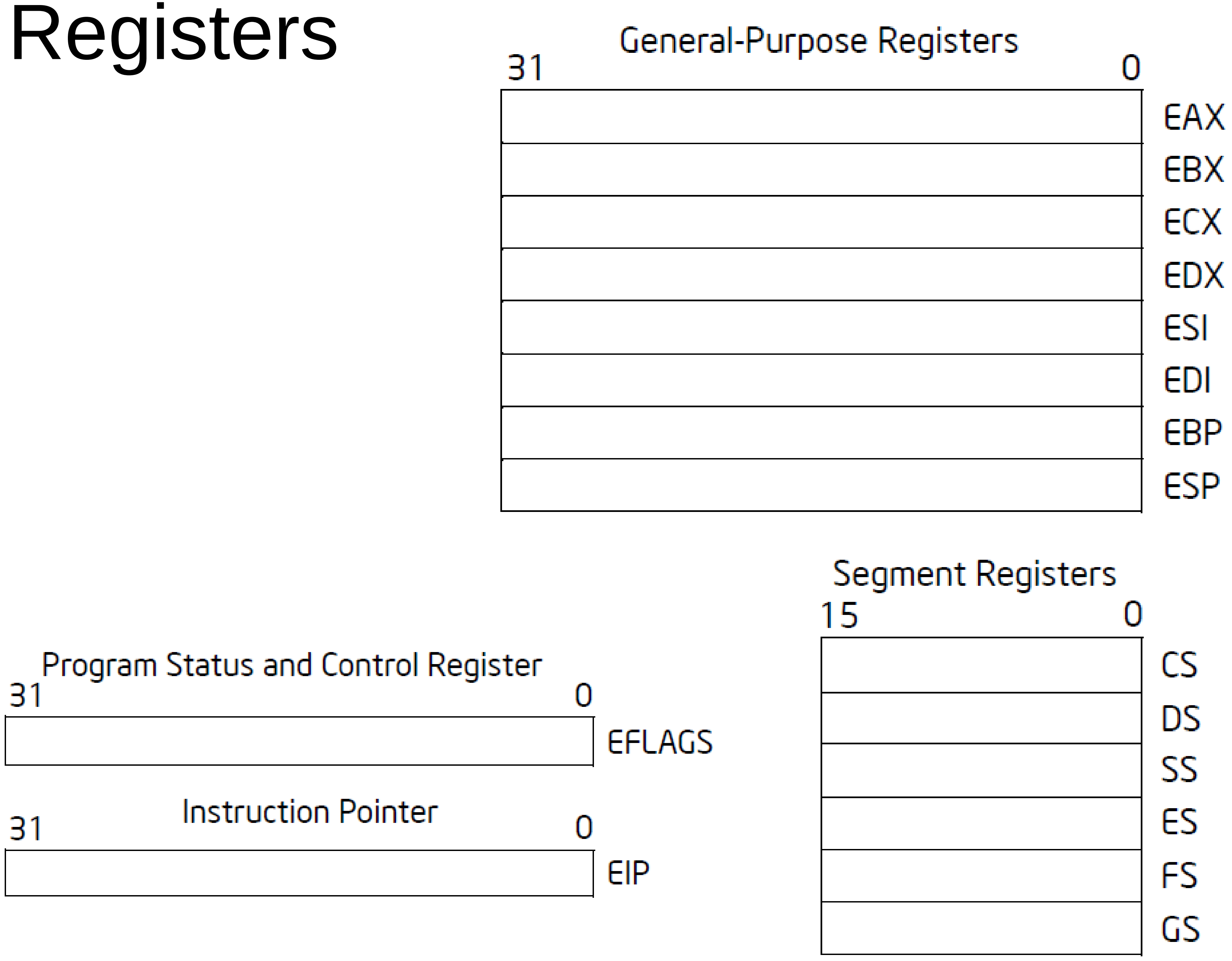
CS5460/6460: Operating Systems

Lecture 4: Hardware interface

Anton Burtsev
January, 2014

Registers



General-Purpose Registers

31 0

EAX

EBX

ECX

EDX

ESI

EDI

EBP

ESP

Segment Registers

15 0

CS

DS

SS

ES

FS

GS

Program Status and Control Register

31 0

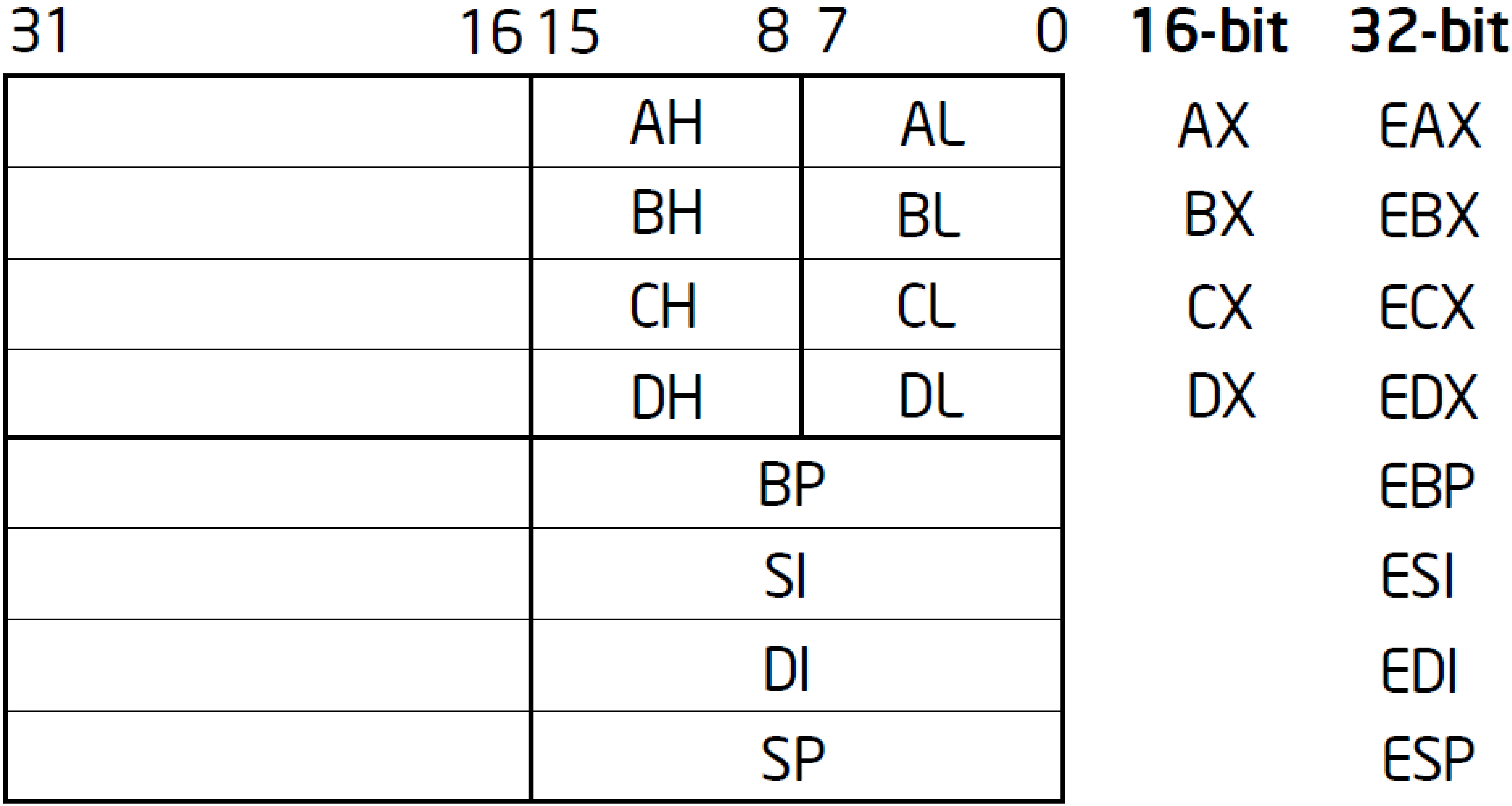
EFLAGS

Instruction Pointer

31 0

EIP

General-Purpose Registers

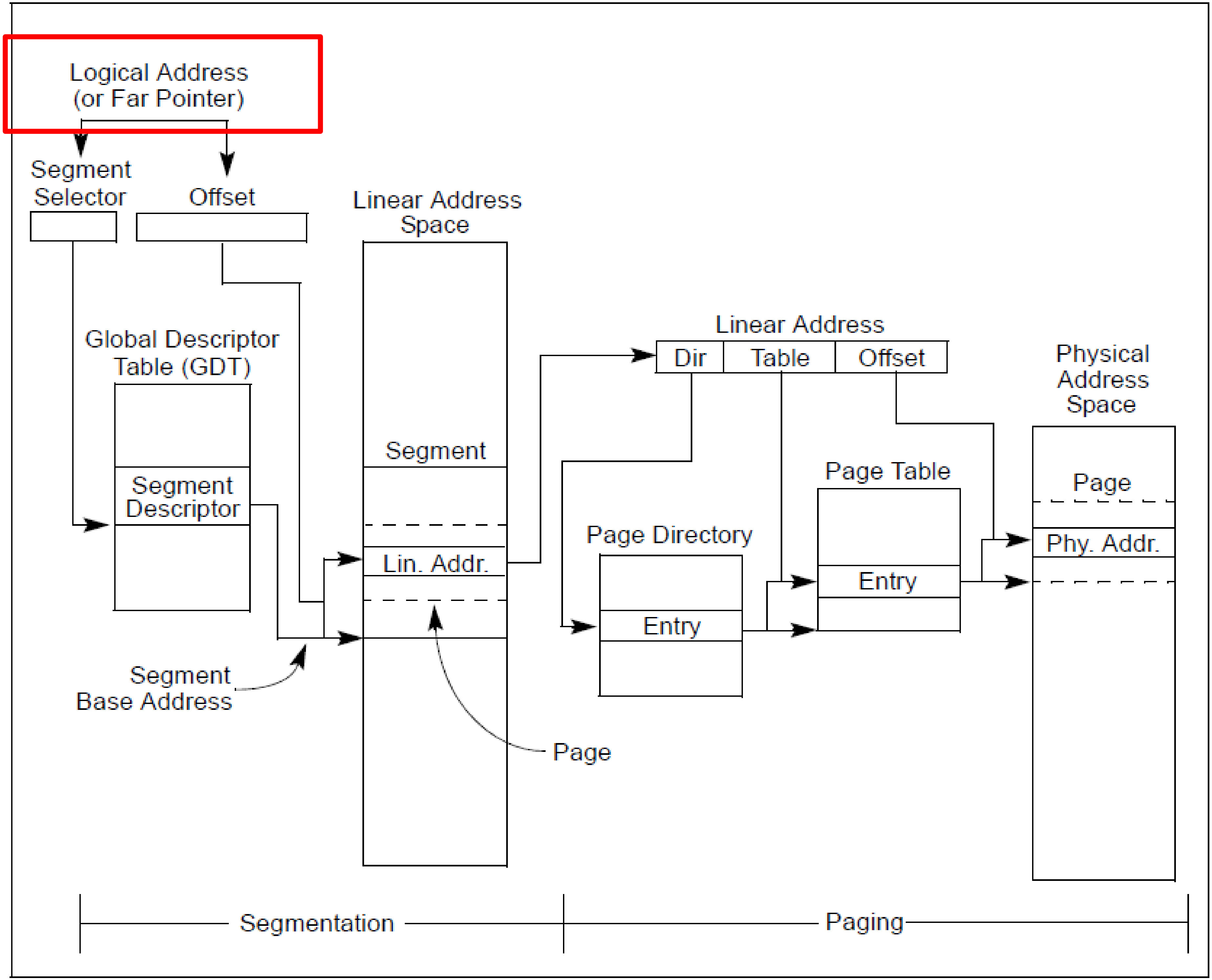


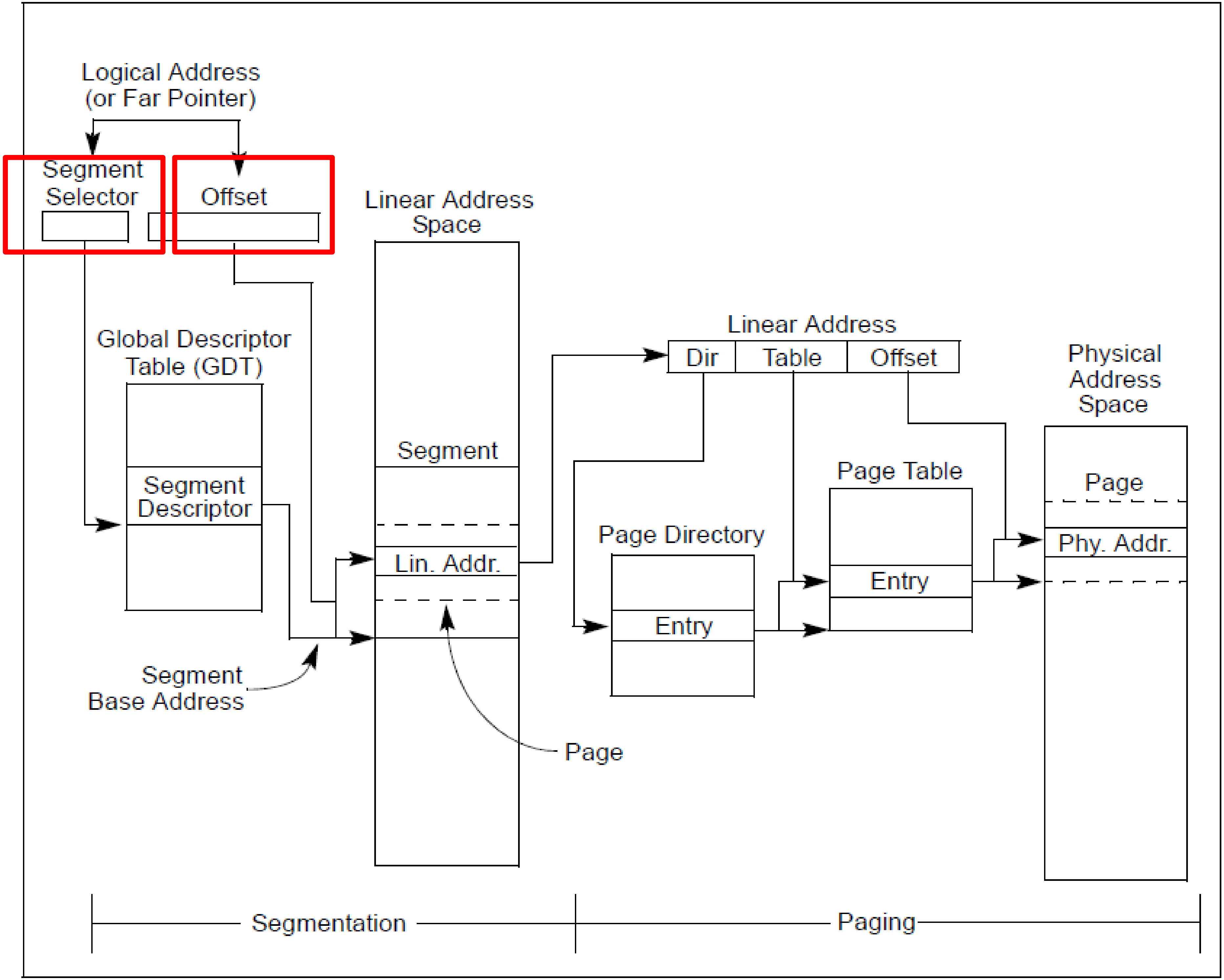
31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

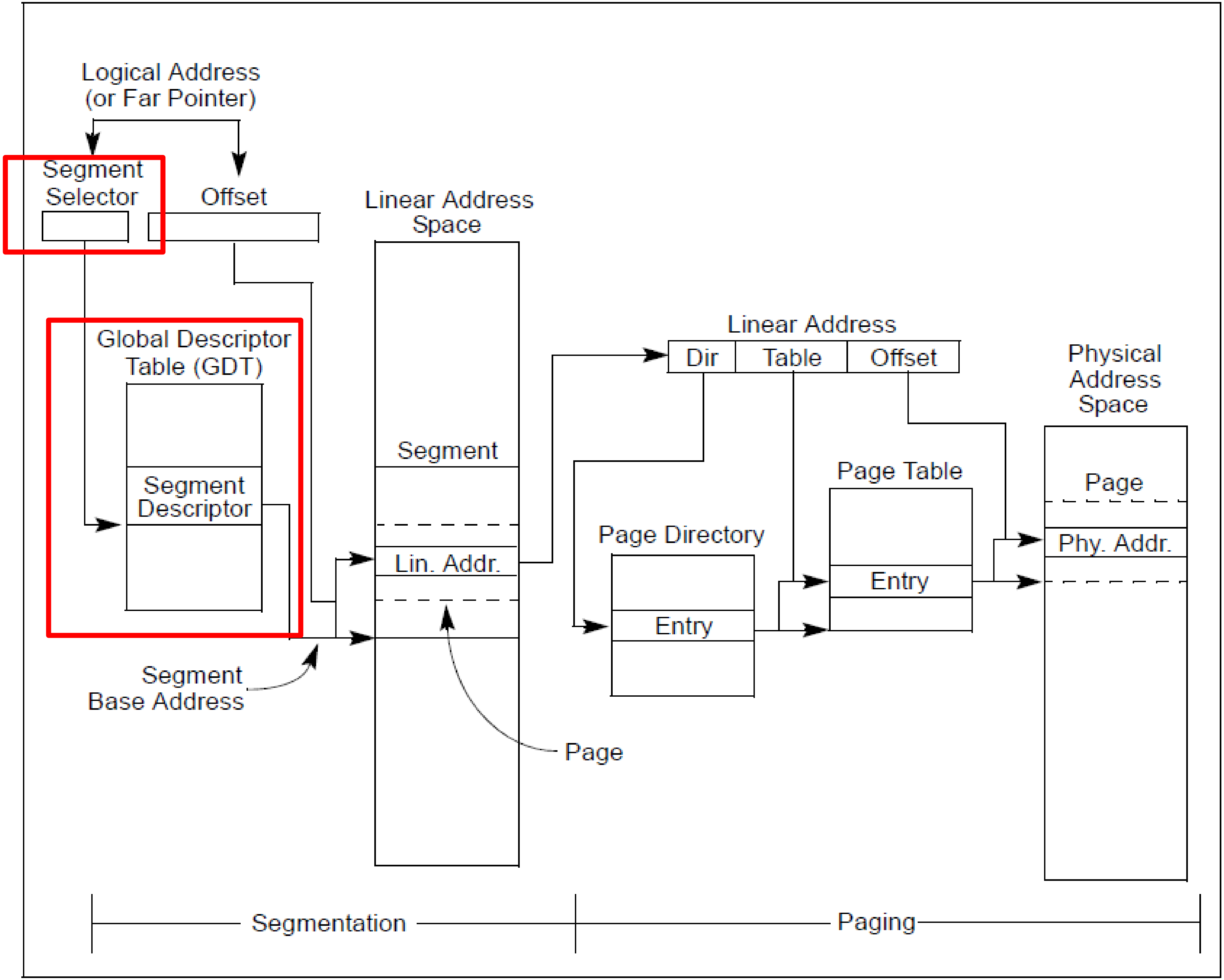
General registers

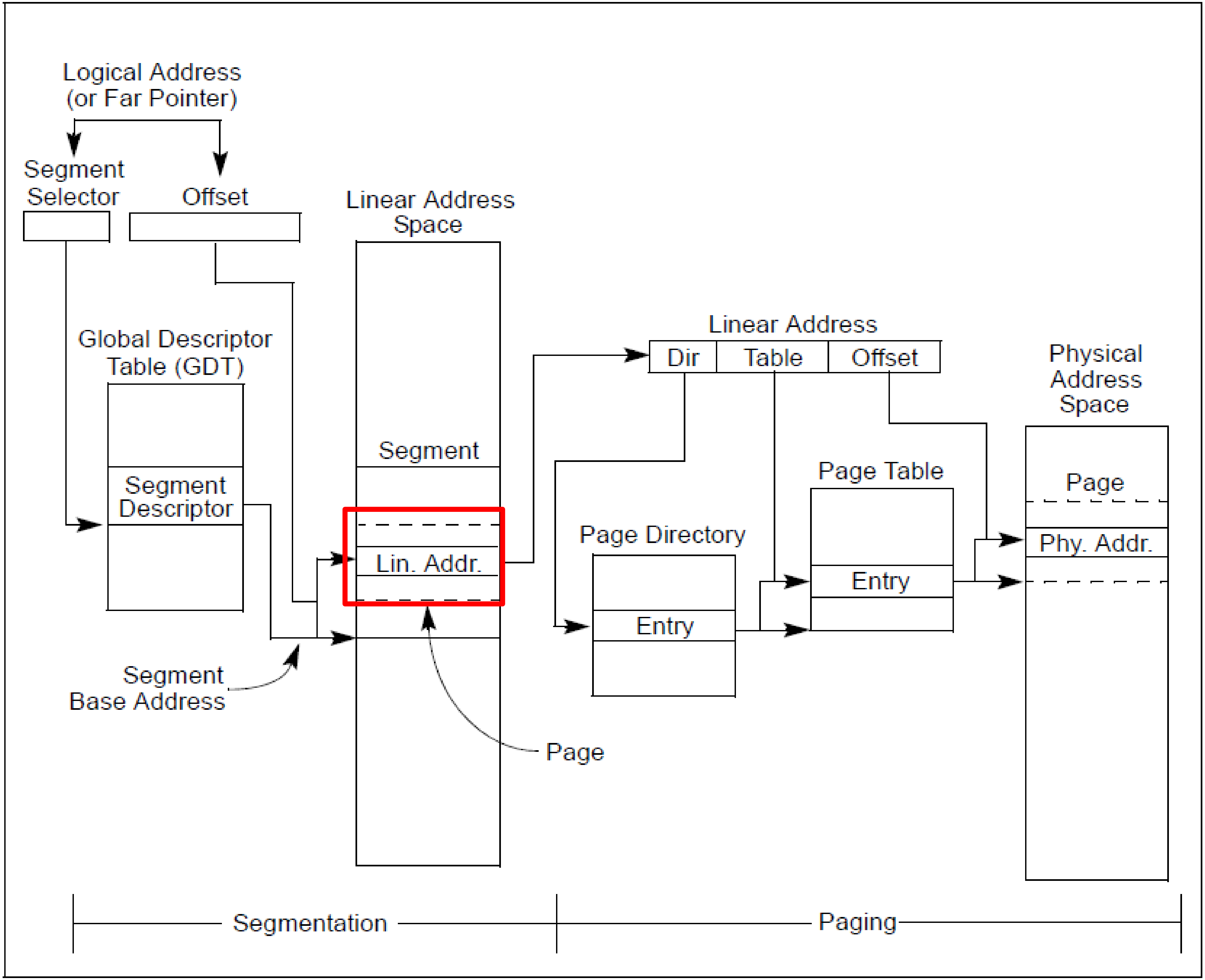
- EAX — Accumulator for operands and results data
- EBX — Pointer to data in the DS segment
- ECX — Counter for string and loop operations
- EDX — I/O pointer
- ESI — Pointer to data in the segment pointed to by the DS register; source pointer for string operations
- EDI — Pointer to data (or destination)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

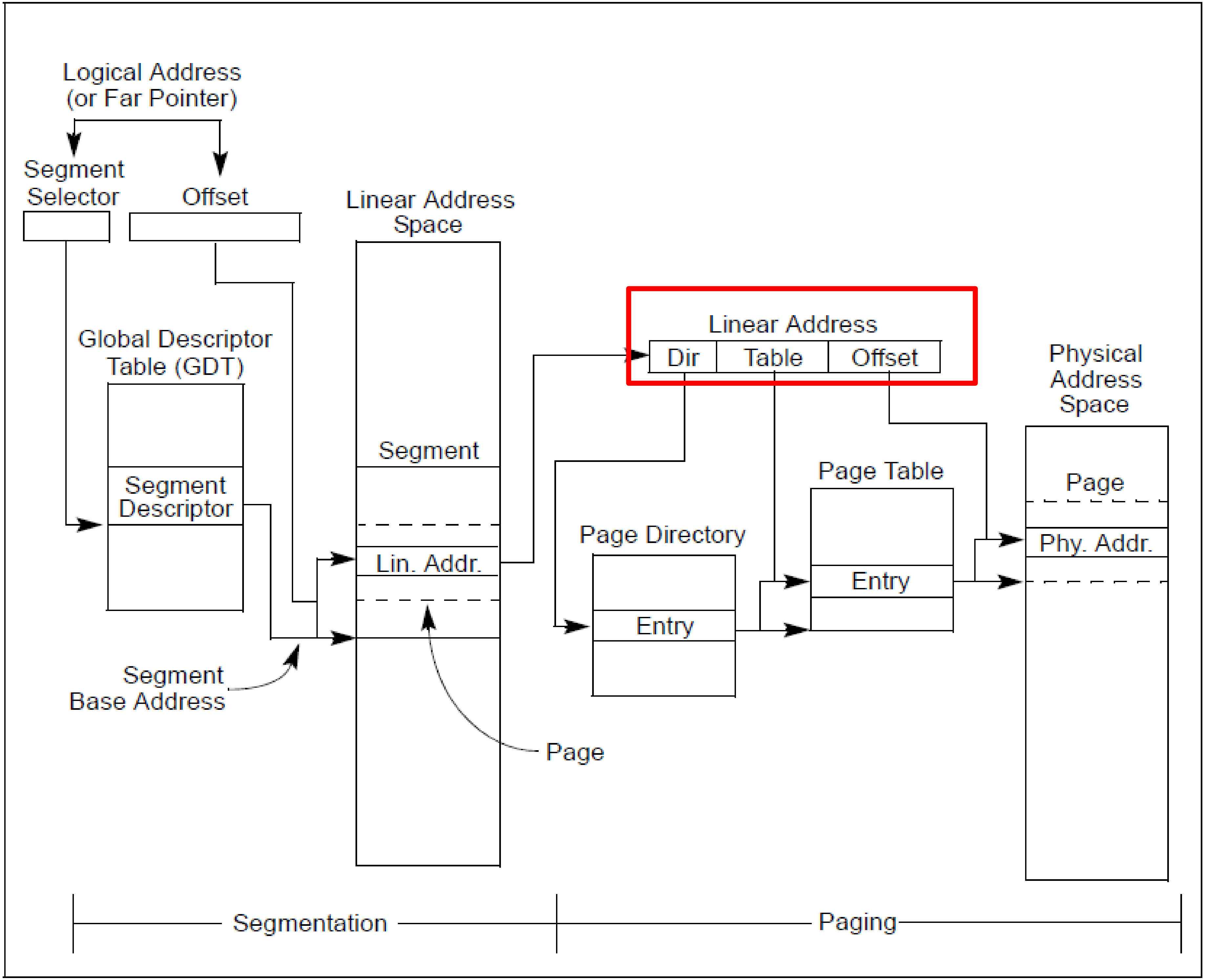
Address translation

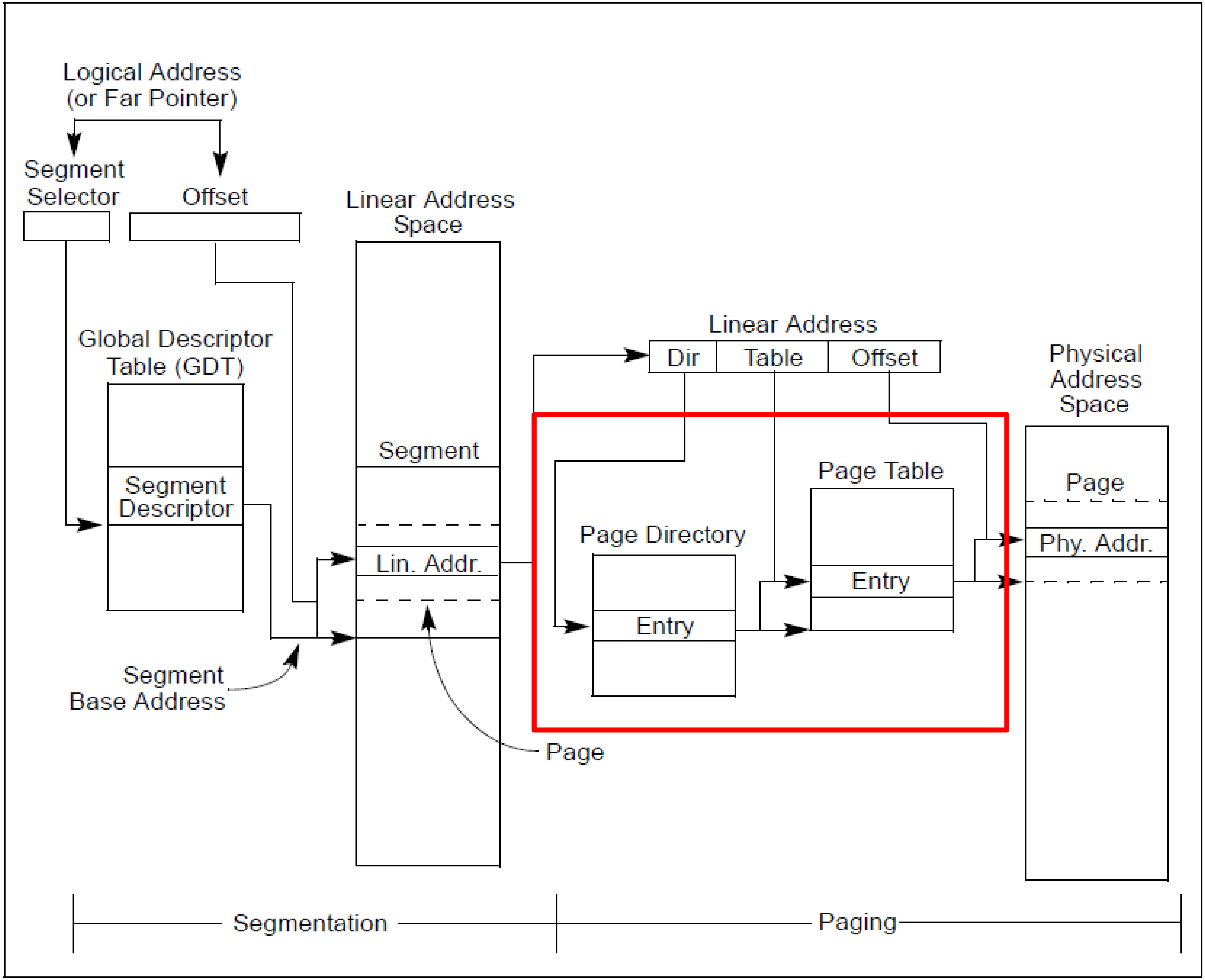

Logical Address
(or Far Pointer)
Segment
Selector
Offset
Linear Address
Space
Global Descriptor
Table (GDT)
Segment
Descriptor
Segment
Lin. Addr.
Segment
Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical
Address
Space
Page
Phy. Addr.
Segmentation
Paging

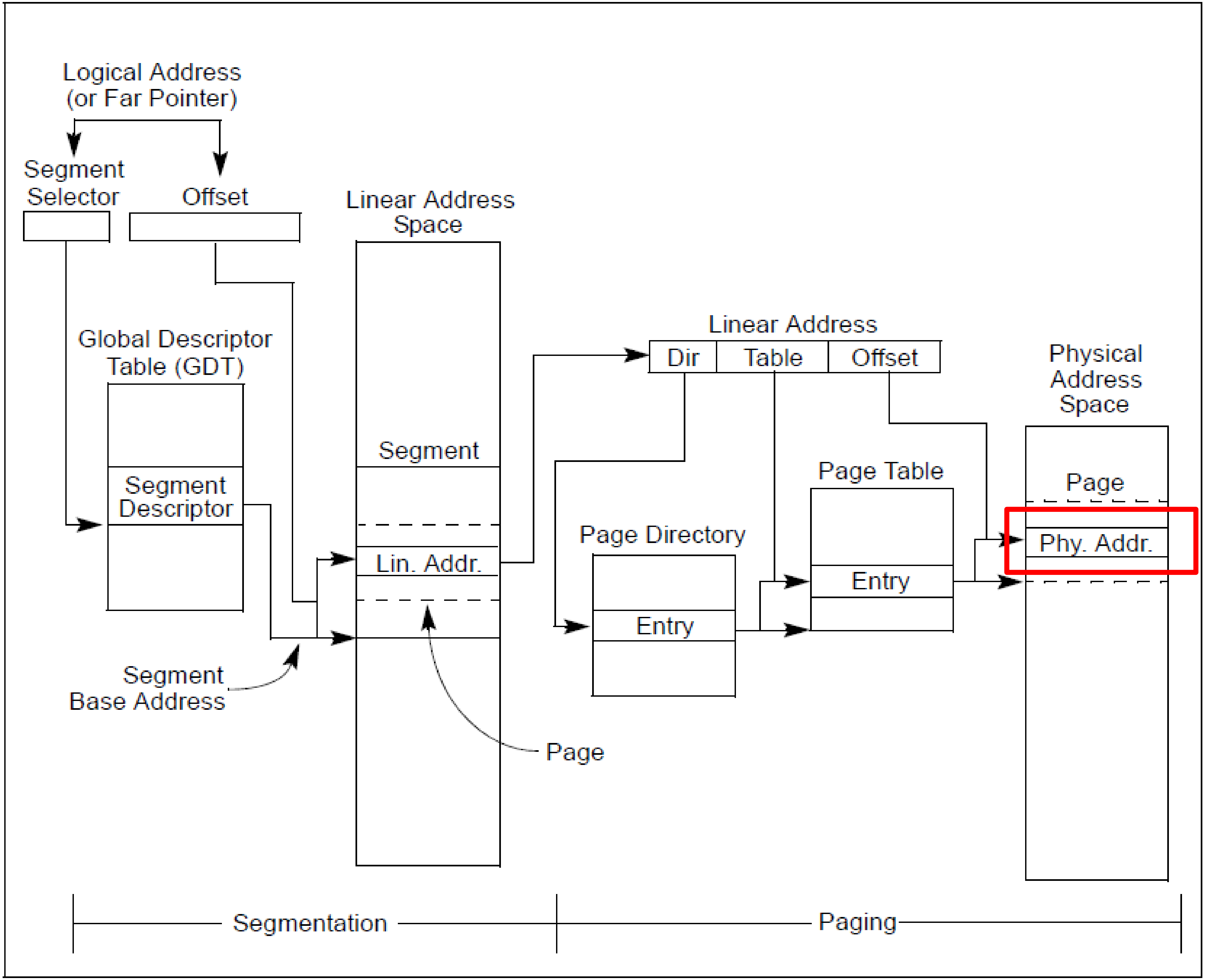

Logical Address
(or Far Pointer)
Segment
Selector
Offset
Linear Address
Space
Global Descriptor
Table (GDT)
Segment
Descriptor
Segment
Lin. Addr.
Segment
Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical
Address
Space
Page
Phy. Addr.
Segmentation
Paging

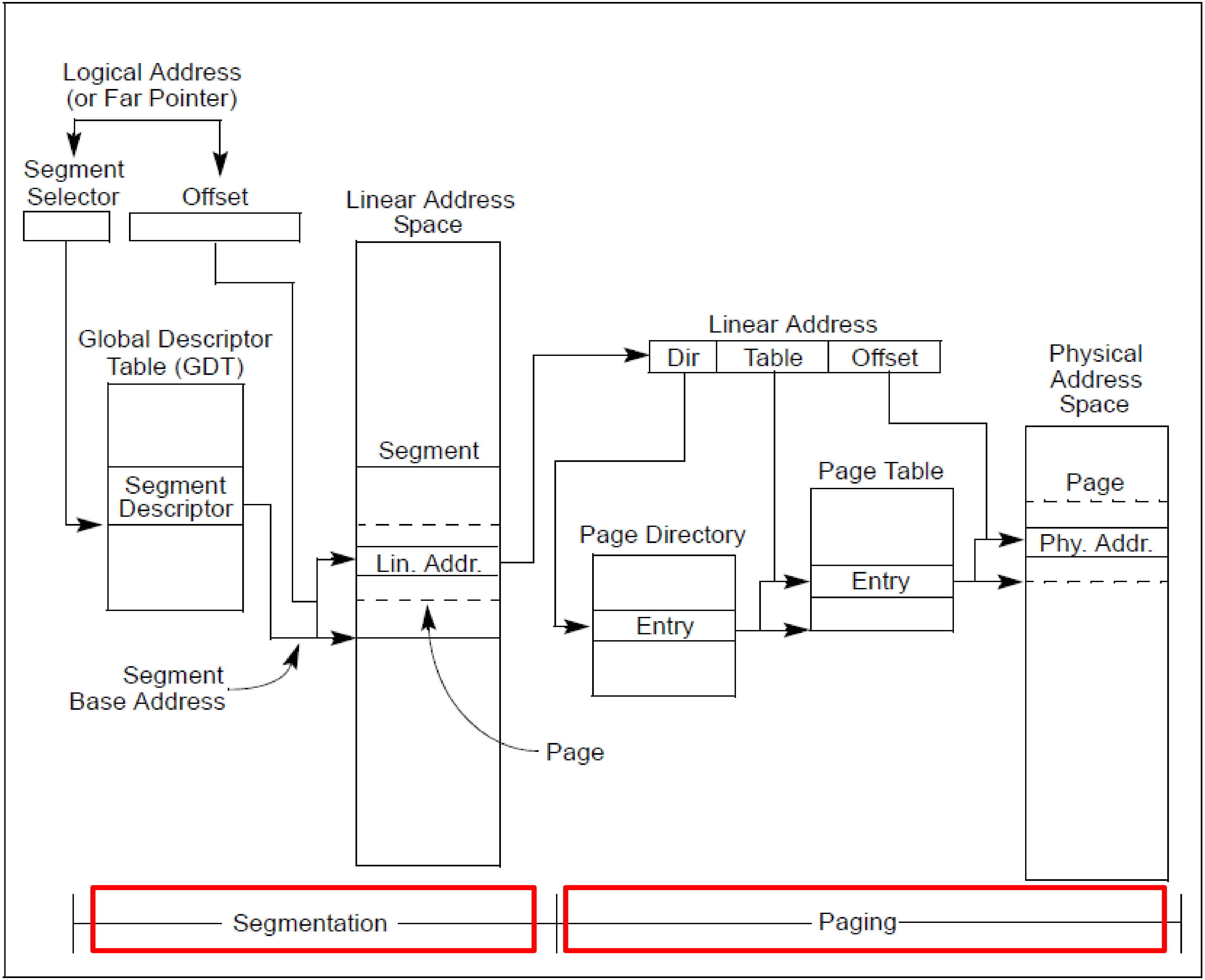

Logical Address
(or Far Pointer)
Segment
Selector
Offset
Linear Address
Space
Global Descriptor
Table (GDT)
Segment
Descriptor
Segment
Lin. Addr.
Segment
Base Address
Page
Linear Address
Dir
Table
Offset
Physical
Address
Space
Page Table
Page Directory
Entry
Entry
Page
Phy. Addr.
Segmentation
Paging


Logical Address
(or Far Pointer)
Segment Selector
Offset
Linear Address Space
Global Descriptor Table (GDT)
Segment Descriptor
Segment
Lin. Addr.
Segment Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical Address Space
Page
Phy. Addr.
Segmentation
Paging


Logical Address
(or Far Pointer)
Segment Selector
Offset
Linear Address Space
Global Descriptor Table (GDT)
Segment Descriptor
Segment
Lin. Addr.
Segment Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical Address Space
Page
Phy. Addr.
Segmentation
Paging


Logical Address
(or Far Pointer)
Segment
Selector
Offset
Linear Address
Space
Global Descriptor
Table (GDT)
Segment
Descriptor
Segment
Lin. Addr.
Segment
Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical
Address
Space
Page
Phy. Addr.
Segmentation
Paging

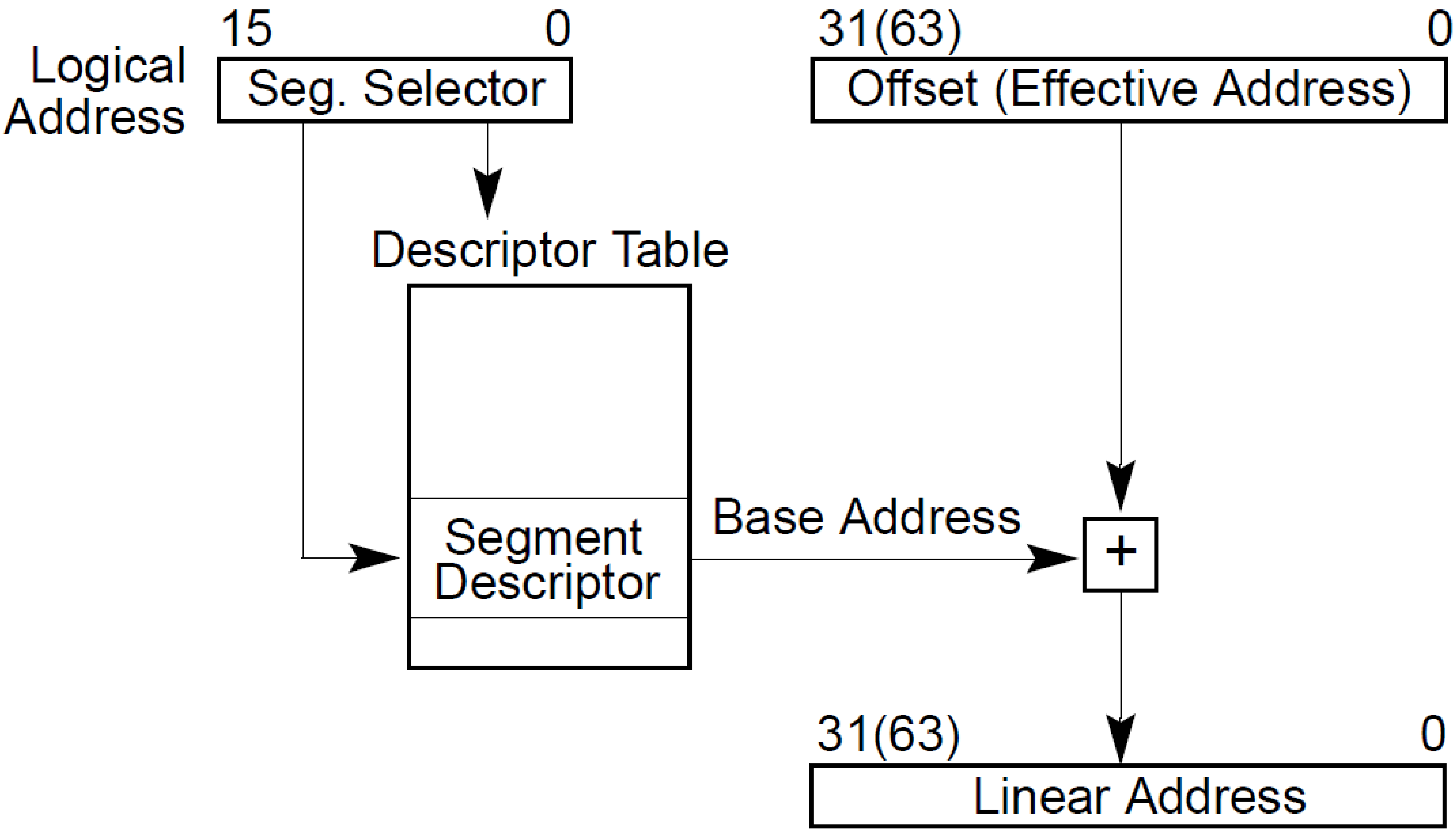

Logical Address
(or Far Pointer)
Segment
Selector
Offset
Linear Address
Space
Global Descriptor
Table (GDT)
Segment
Descriptor
Segment
Lin. Addr.
Segment
Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical
Address
Space
Page
Phy. Addr.
Segmentation
Paging


Logical Address
(or Far Pointer)
Segment Selector
Offset
Linear Address Space
Global Descriptor Table (GDT)
Segment Descriptor
Segment
Lin. Addr.
Segment Base Address
Page
Linear Address
Dir
Table
Offset
Page Directory
Entry
Page Table
Entry
Physical Address Space
Page
Phy. Addr.
Segmentation
Paging

Segmentation

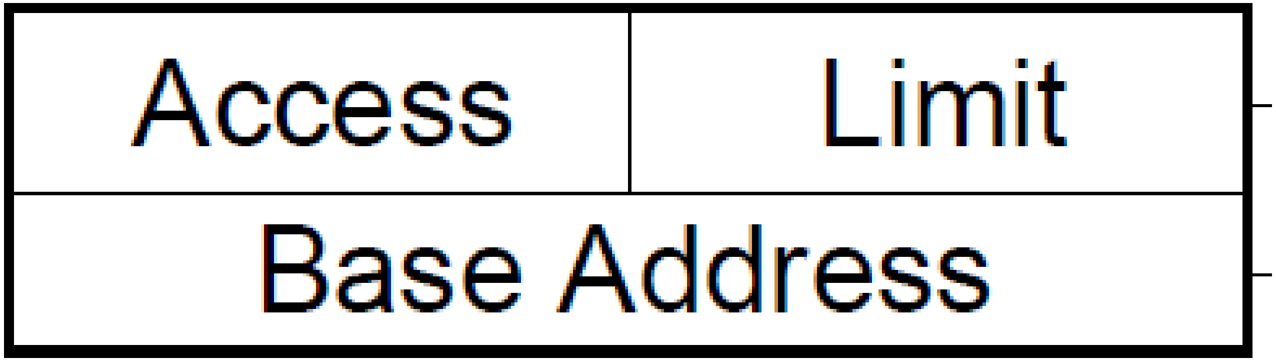
Logical address

- Segment selector (16 bit) + offset (32 bit)



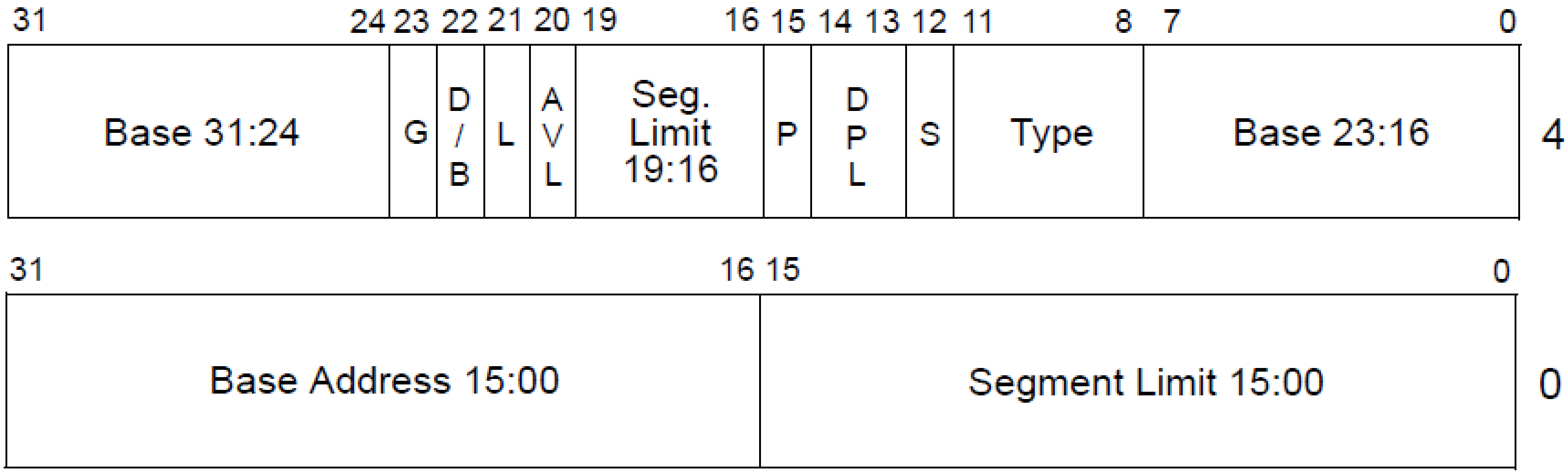
Segment descriptors

- Base address
  - 0 – 4 GB
- Limit (size)
  - 0 – 4 GB
- Access rights
  - Executable, readable, writable
  - Privilege level (0 - 3)



Access	Limit
Base Address	

Segment descriptors



31 24	23	22	21	20	19 16	15	14 13	12	11 8	7 0	
Base 31:24	G	D/B	L	AVL	Seg. Limit 19:16	P	DPL	S	Type	Base 23:16	4

31 16	15 0	
Base Address 15:00	Segment Limit 15:00	0

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

Segment registers

- Hold 16 bit segment selectors
  - Pointers into a special table
  - Global or local descriptor table
- Segments are associated with one of three types of storage
  - Code
  - Data
  - Stack

Code segment

- Code
  - CS register
  - EIP is an offset inside the segment stored in CS
- Can only be changed with
  - procedure calls,
  - interrupt handling, or
  - task switching

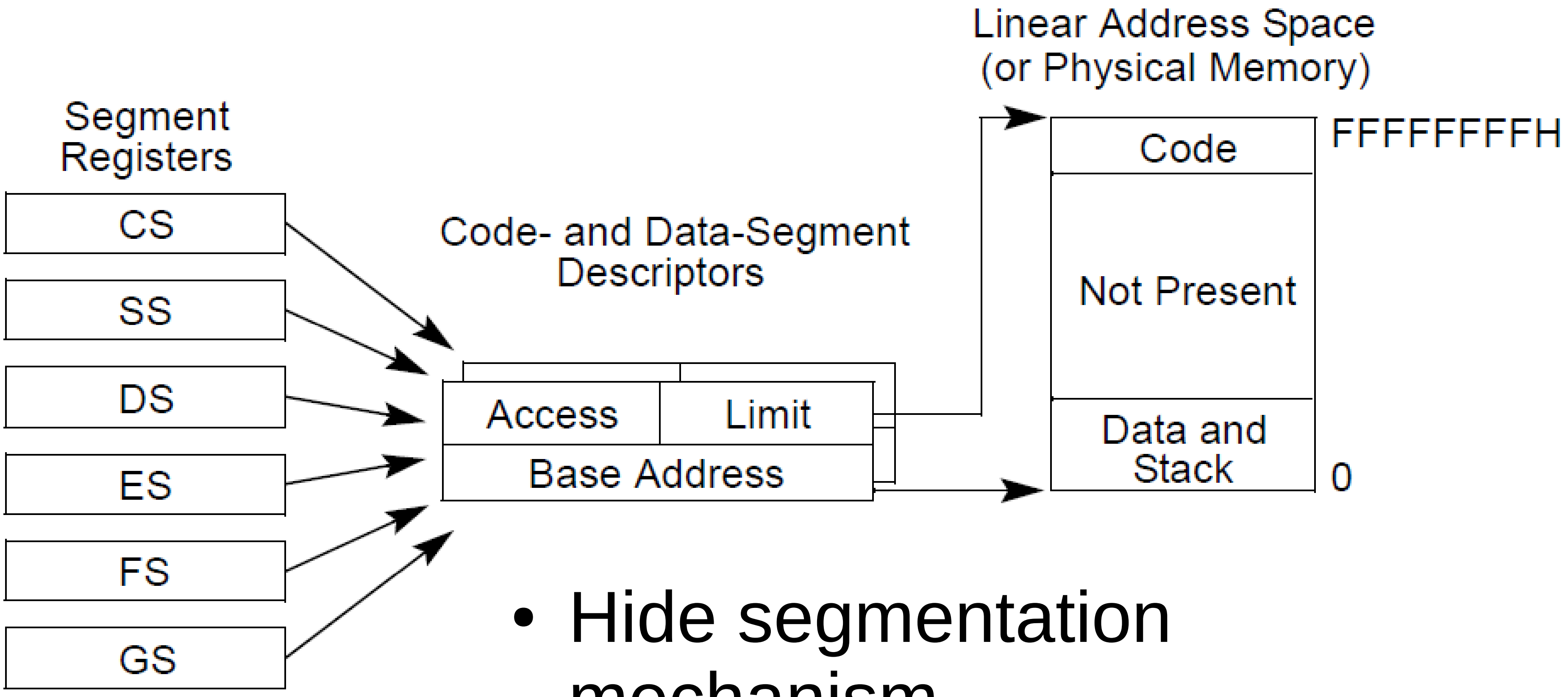
Data segment

- Data
  - DS, ES, FS, GS
  - 4 possible data segments can be used at the same time

Stack segment

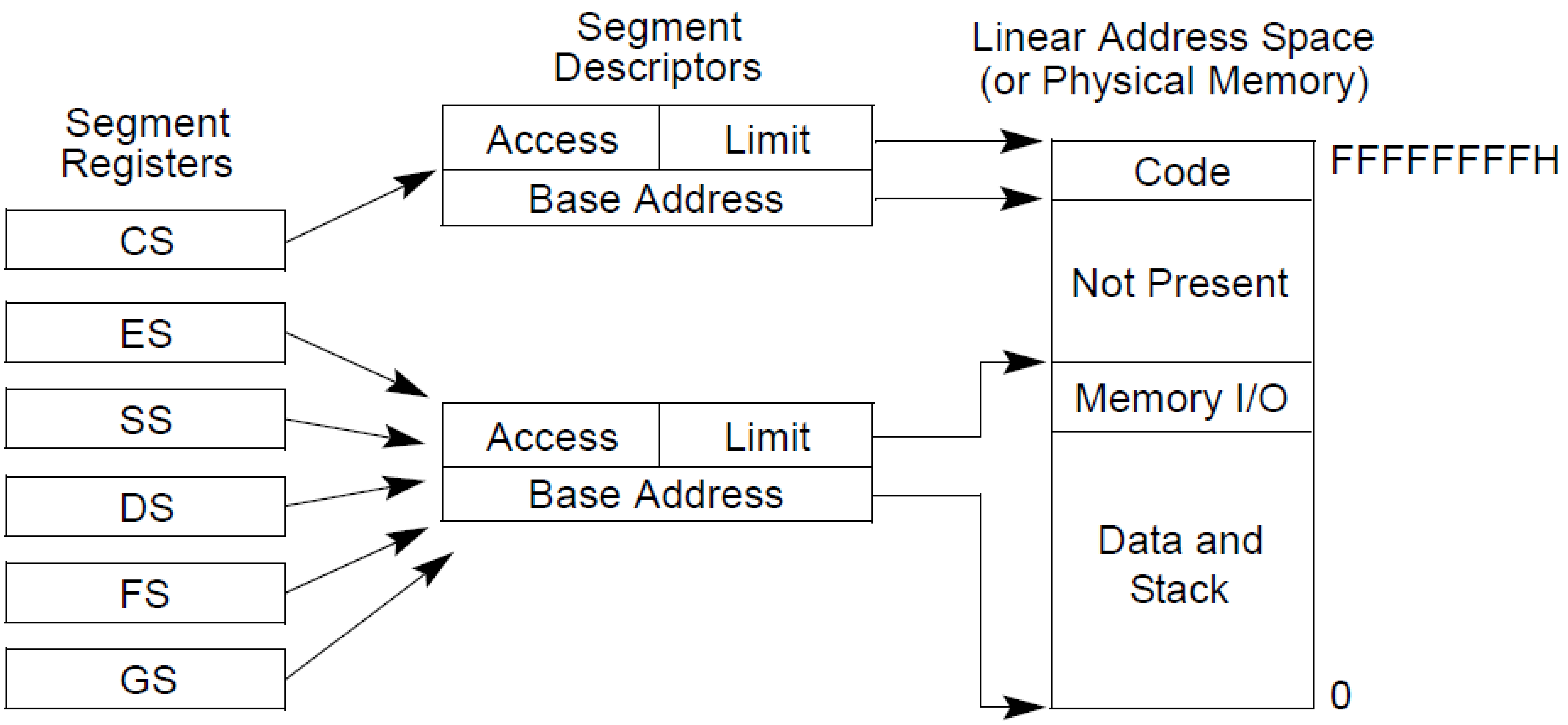
- Stack
  - SS
- Can be loaded explicitly
  - OS can set up multiple stacks
  - Of course, only one is accessible at a time

Flat model

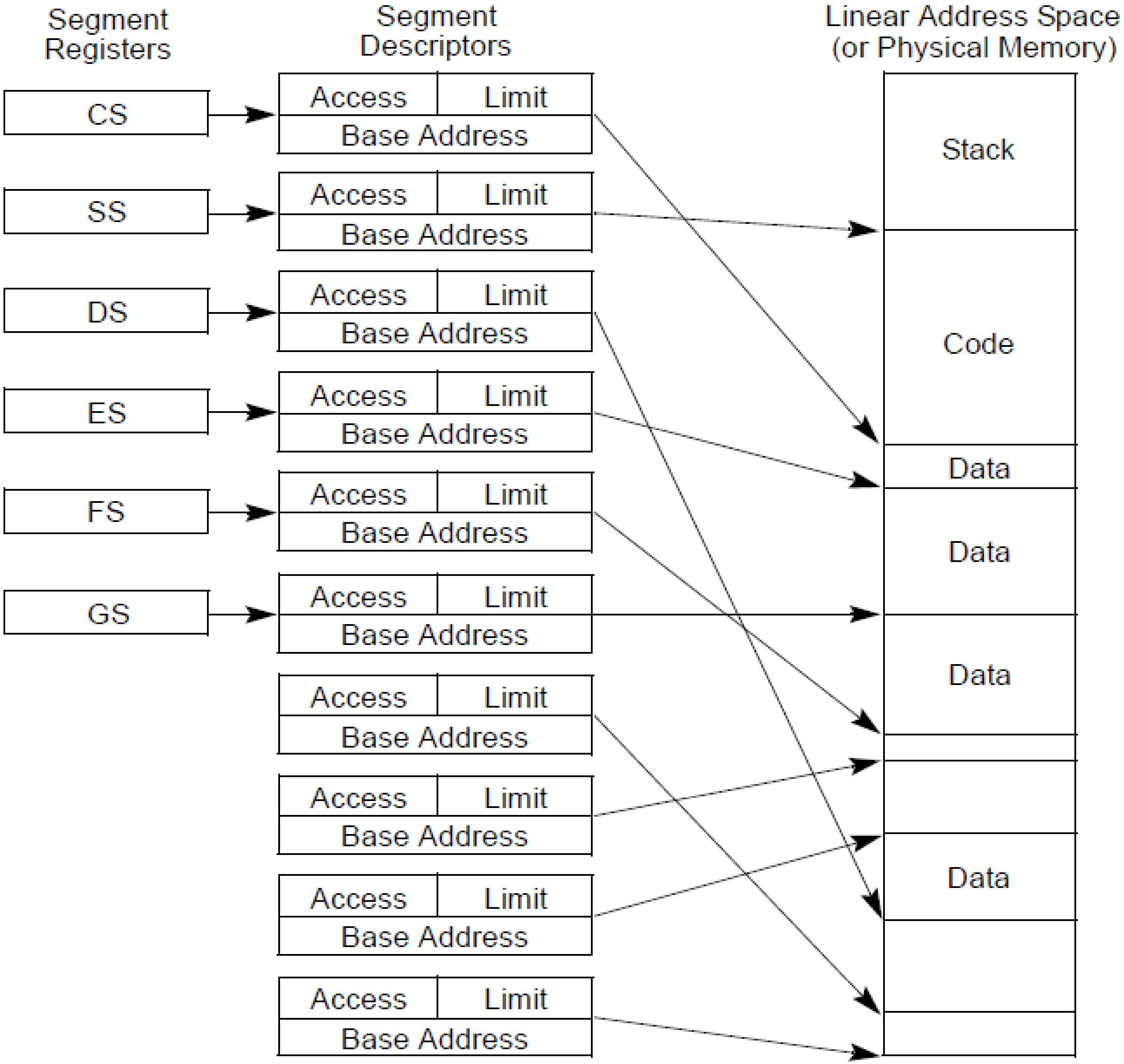


- Hide segmentation mechanism
- But allows access to nonexistent memory

Protected flat



Multi-Segment

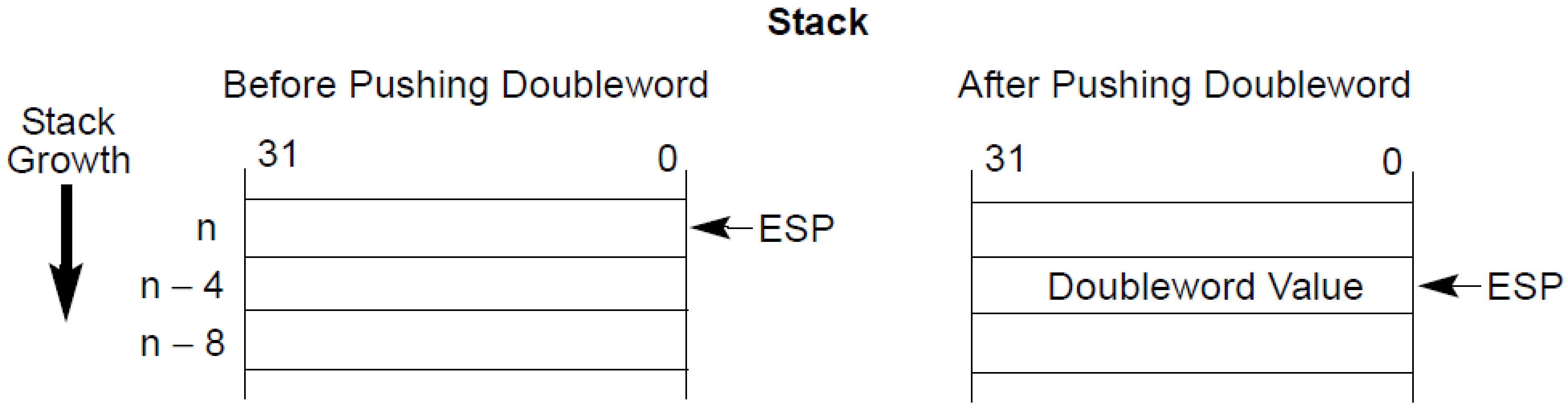


Stack

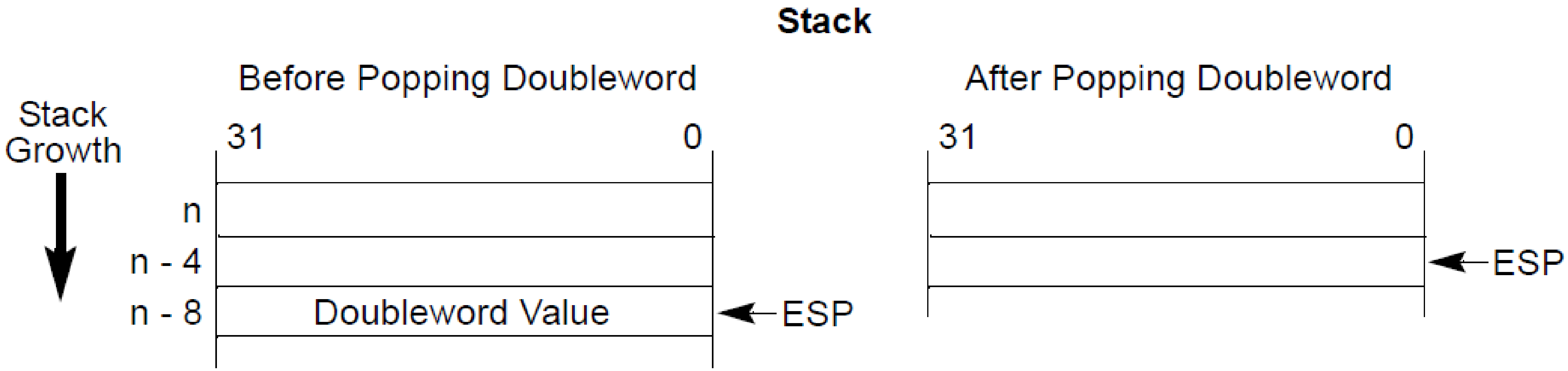
Stack

- SS
  - Specifies stack segment
- ESP
  - Contains the address of the data that would be removed from the stack
- PUSH/POP
  - Insert/remove data on the stack
  - Subtract/add 4 to ESP

Example: PUSH



Example: POP

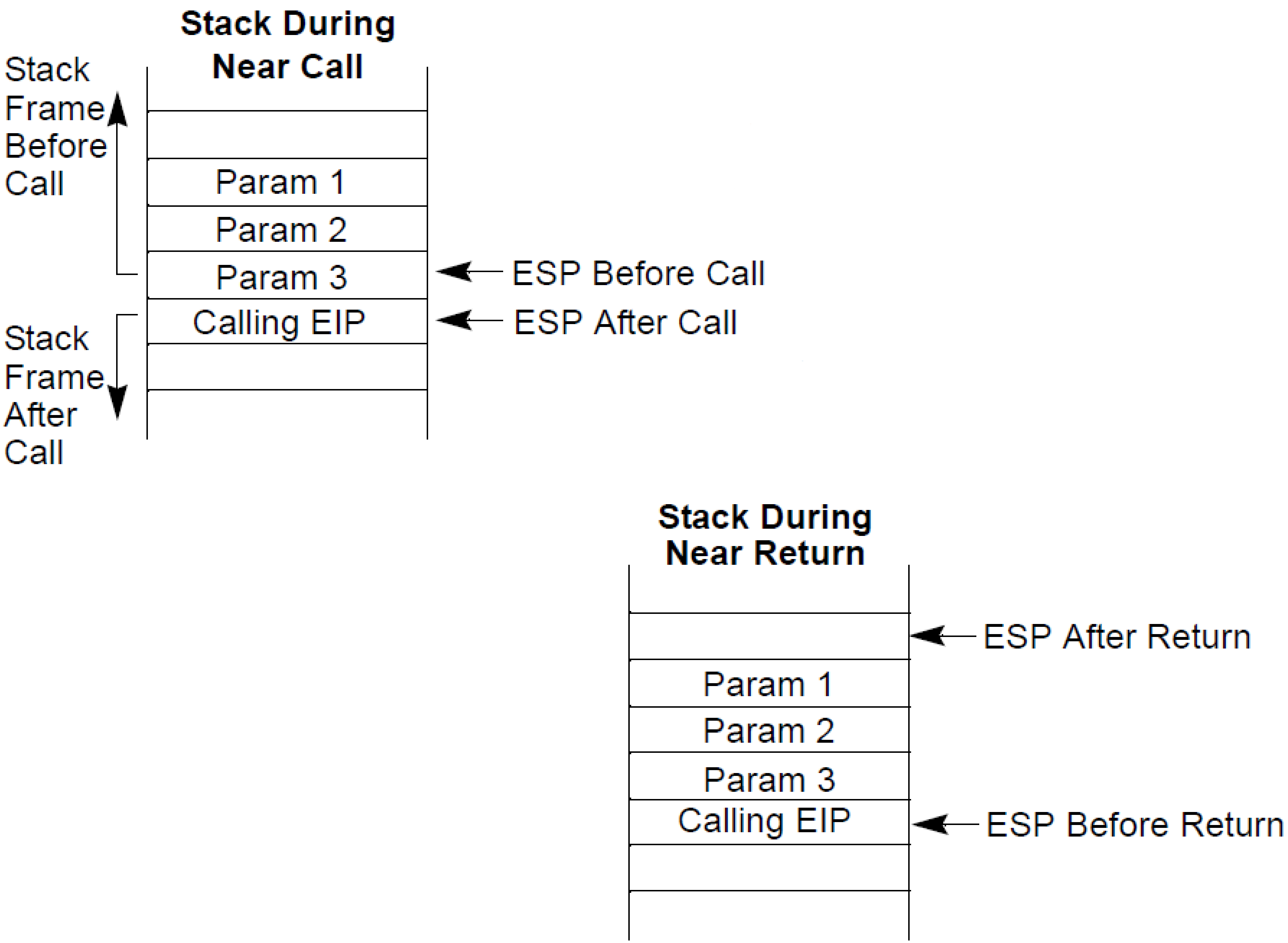

Stack
Before Popping Doubleword
After Popping Doubleword
Stack Growth
31
0
n
n - 4
n - 8
Doubleword Value
ESP
31
0
ESP

Setting up stack

- Create a stack descriptor
  - Base, limit, access rights
- Load stack selector into SS register
  - MOV, POP, or LSS
- Load the stack pointer into ESP
  - MOV, POP, or LSS

Call/return

- Stack is used to implement function invocations
- CALL
  - Makes an unconditional jump to a subprogram and pushes the address of the next instruction on the stack
- RET
  - Pops off an address and jumps to that address



Stack During Near Call

Stack Frame Before Call

Param 1

Param 2

Param 3

ESP Before Call

Calling EIP

ESP After Call

Stack Frame After Call

Stack During Near Return

ESP After Return

Param 1

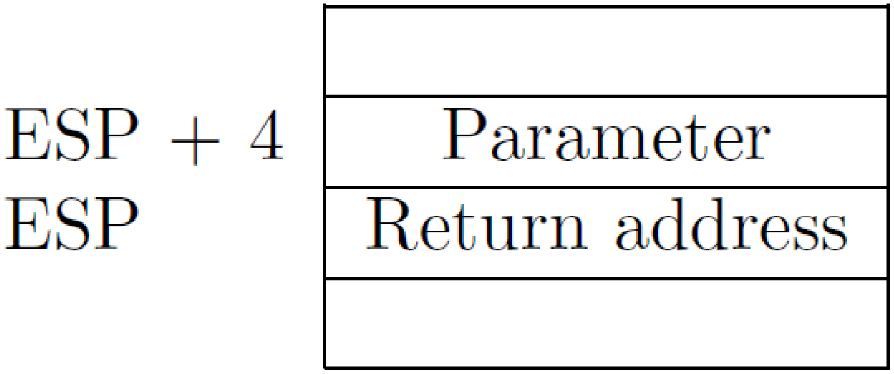
Param 2

Param 3

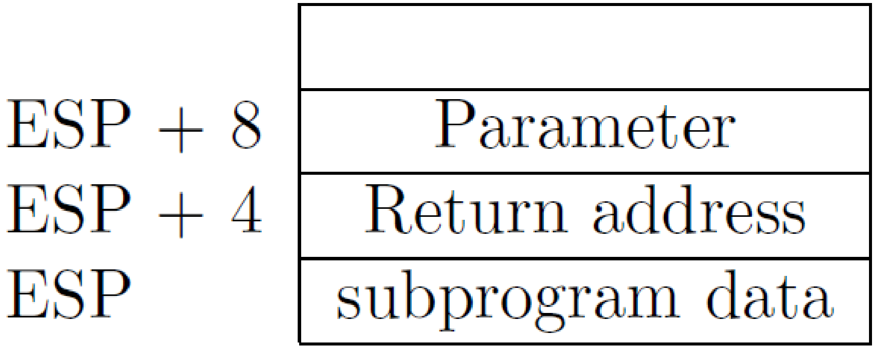
Calling EIP

ESP Before Return

Stack bottom pointer



ESP + 4	Parameter
ESP	Return address



ESP + 8	Parameter
ESP + 4	Return address
ESP	subprogram data

Initially parameter is

- [ESP + 4]

Later as the function pushes things on the stack it changes, e.g.

- [ESP + 8]

- Use dedicated register **EBP**

Prologue/epilogue

```
subprogram_label:
    push ebp          ; save original EBP value on stack
    mov ebp, esp      ; new EBP = ESP
; subprogram code
    pop ebp           ; restore original EBP value
    ret
```

- Example invocation

```
    push dword 1      ; pass 1 as parameter
    call fun
    add esp, 4        ; remove parameter from stack
```

Calling conventions

- Goal: reentrant programs
    - Conventions differ from compiler, optimizations, etc.
- Call/return are used for function invocations
- Parameters passed on the stack
    - Pushed onto the stack before the CALL instruction

Local variables

- Stored right after the saved EBP value in the stack
- Allocated by subtracting the number of bytes required from ESP

```
subprogram_label:
    push ebp                          ; save original EBP value on stack
    mov ebp, esp                      ; new EBP = ESP
    sub esp, LOCAL_BYTES ; = # bytes needed by locals
; subprogram code
    mov esp, ebp                      ; deallocate locals
    pop ebp                           ; restore original EBP value
    ret
```

Parameter passing

- Registers
- On the stack
- Through memory
  - Pass a pointer to the parameter list in one of the registers

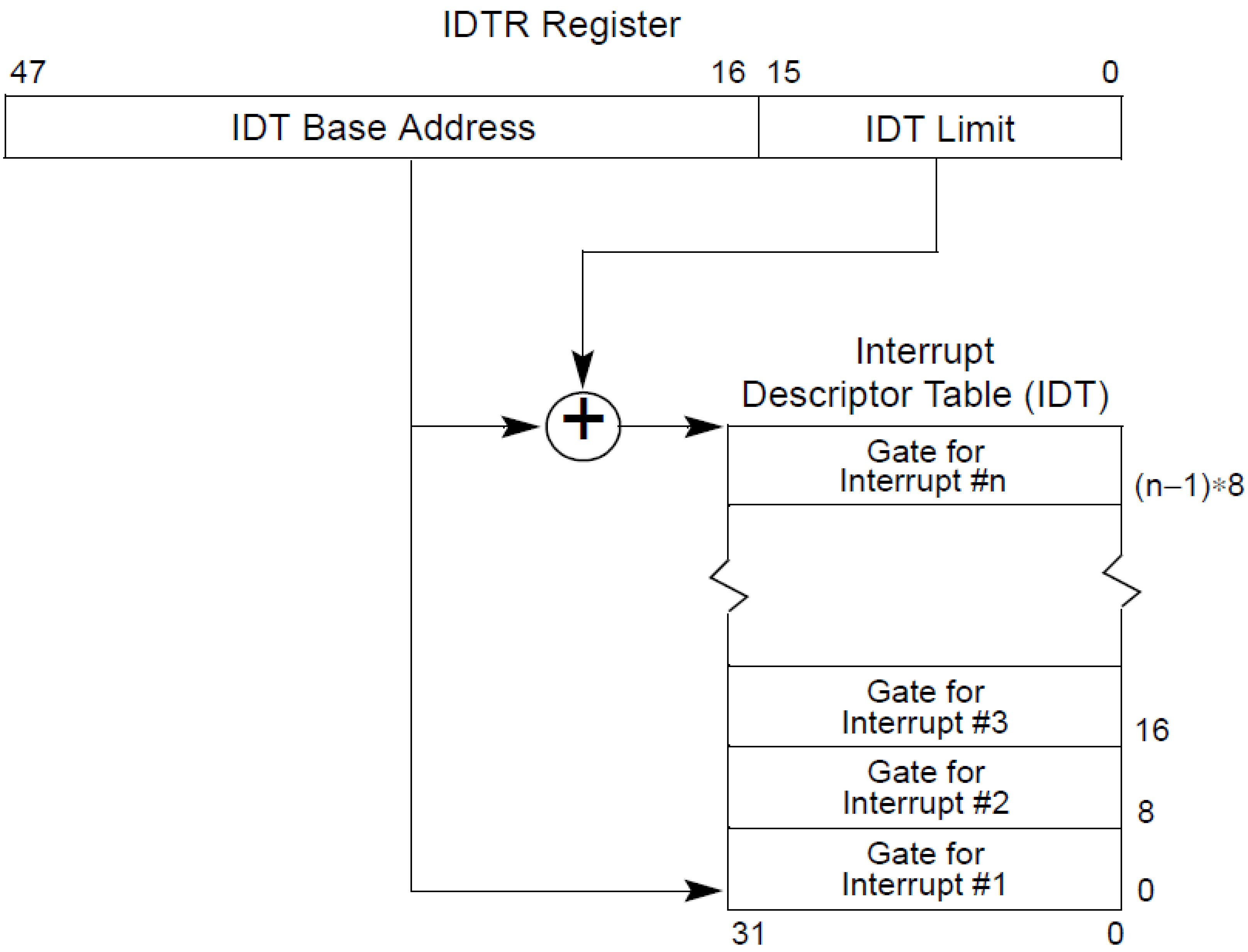
Saving state

- Processor doesn't save registers
  - General purpose, segment, flags
- Calling convention is needed
  - Agreement on what gets saved by a callee and caller

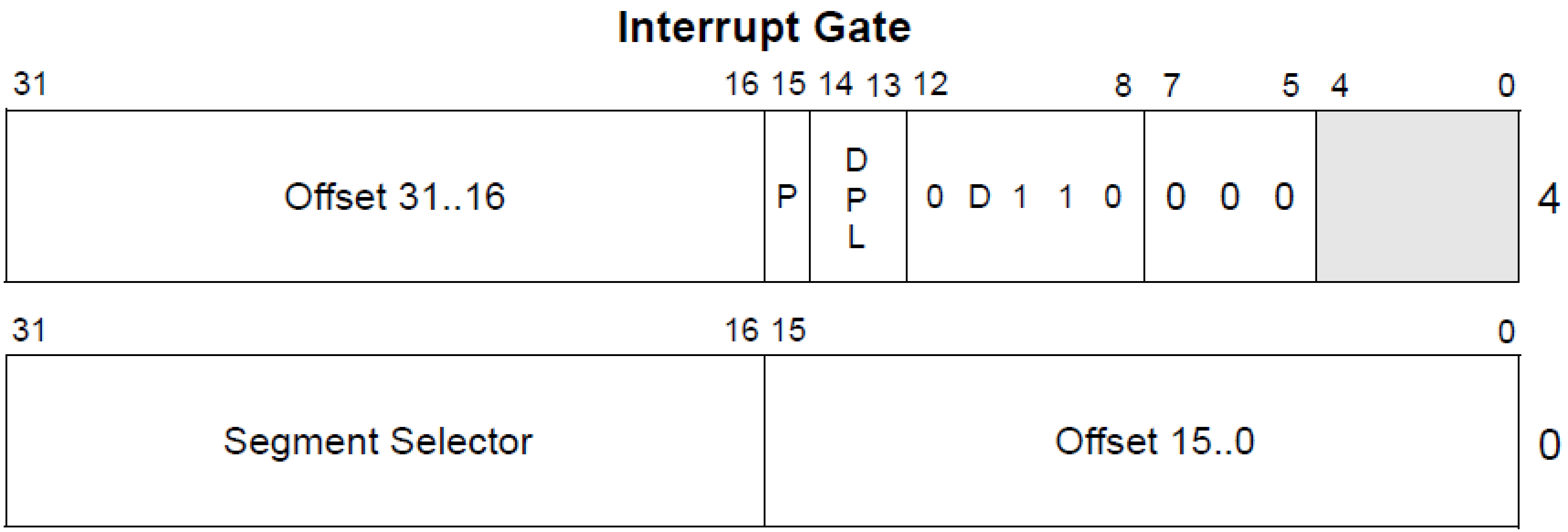
Interrupts

INT X

- Transfers control to the handler number X in a special table
  - Interrupt descriptor table
- IDT can be anywhere in the linear address space
  - Located with the IDTR register


IDTR Register
47
16
15
0
IDT Base Address
IDT Limit
+
Interrupt
Descriptor Table (IDT)
Gate for
Interrupt #n
(n–1)*8
Gate for
Interrupt #3
16
Gate for
Interrupt #2
8
Gate for
Interrupt #1
0
31
0

Interrupt descriptor



Interrupt Gate

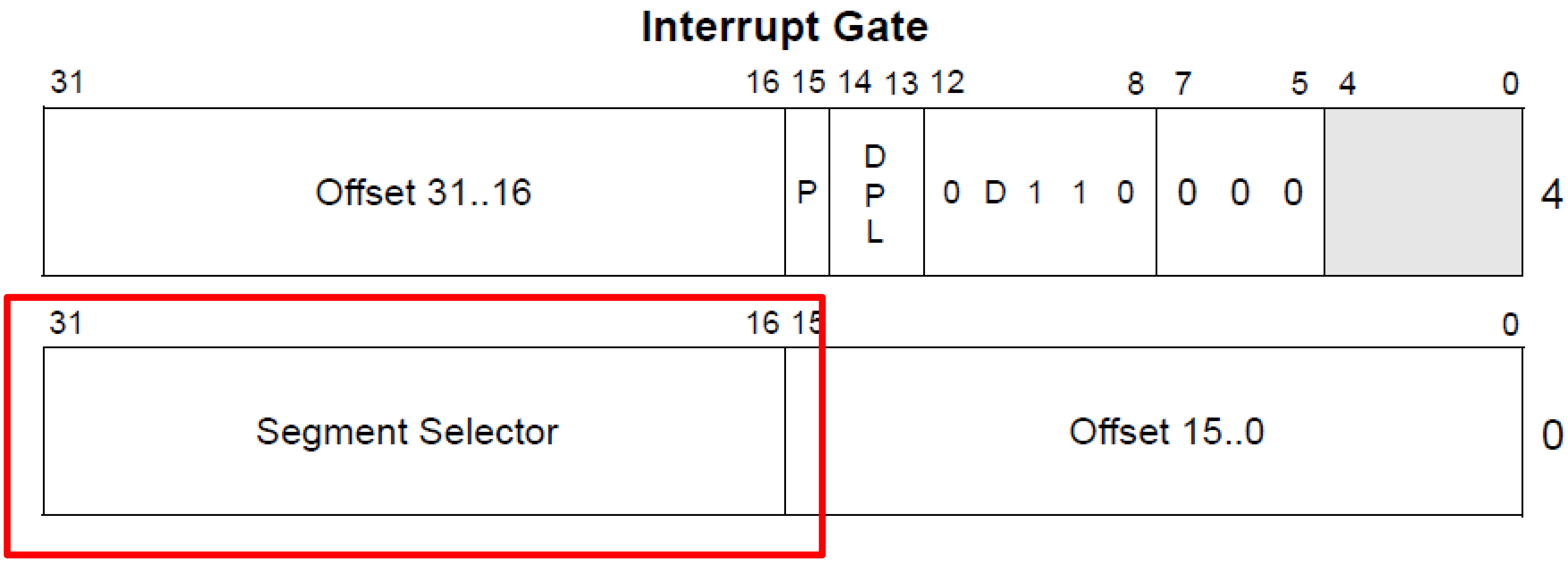
31 16 15 14 13 12 8 7 5 4 0

Offset 31..16 | P | DPL | 0 D 1 1 0 | 0 0 0 | 4

31 16 15 0

Segment Selector | Offset 15..0 | 0

Interrupt descriptor



Interrupt Gate

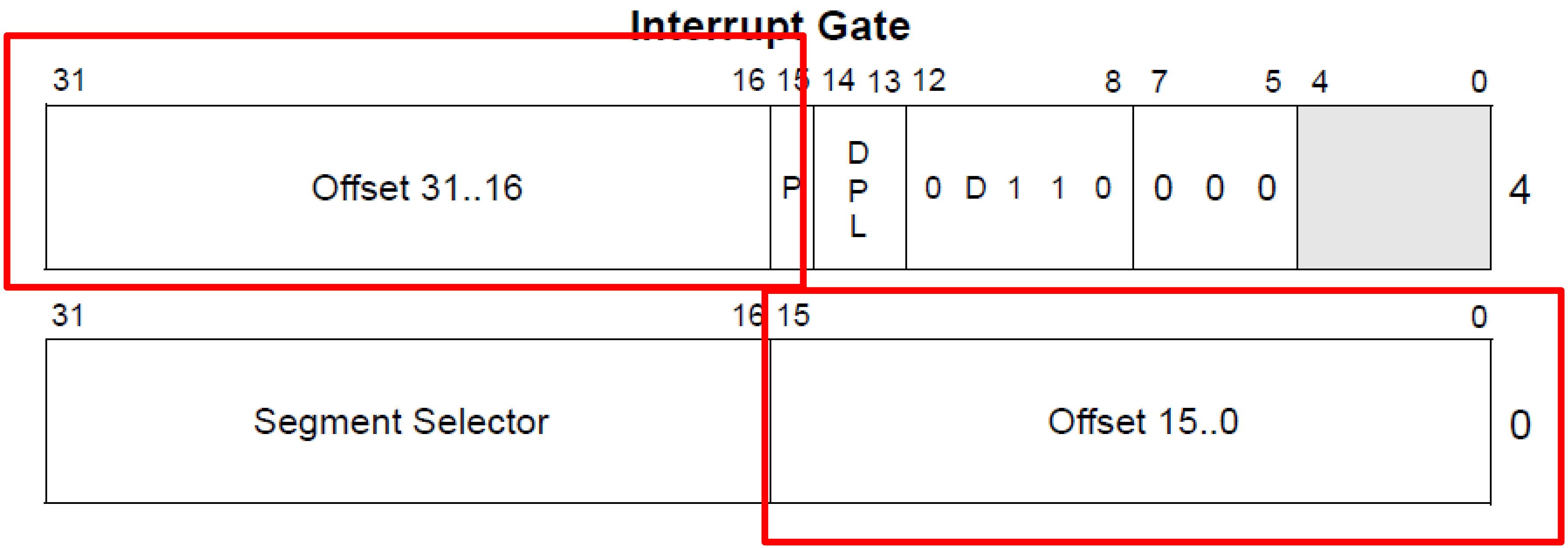
31 16 15 14 13 12 8 7 5 4 0

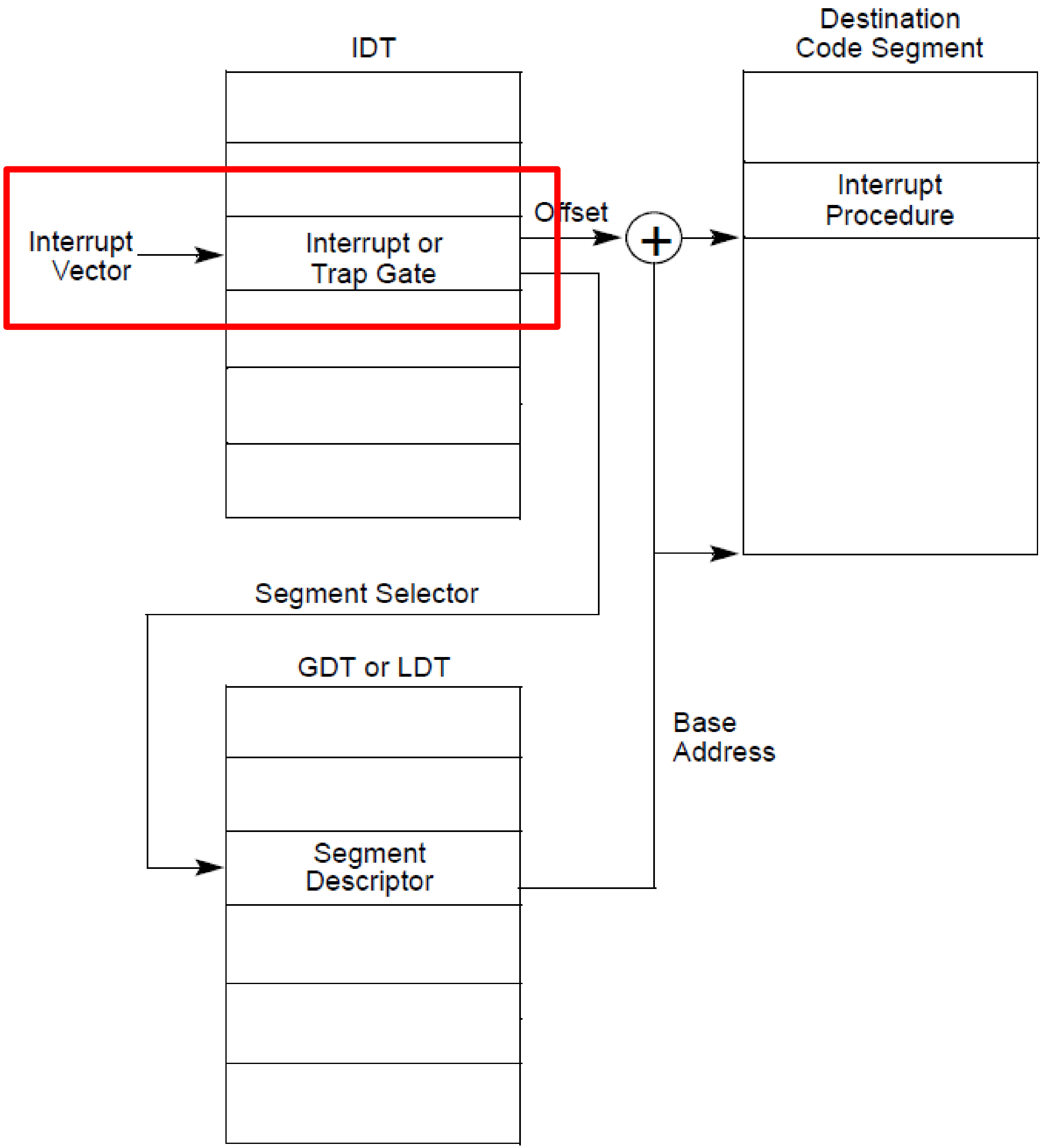
Offset 31..16 | P | DPL | 0 D 1 1 0 | 0 0 0 | | 4

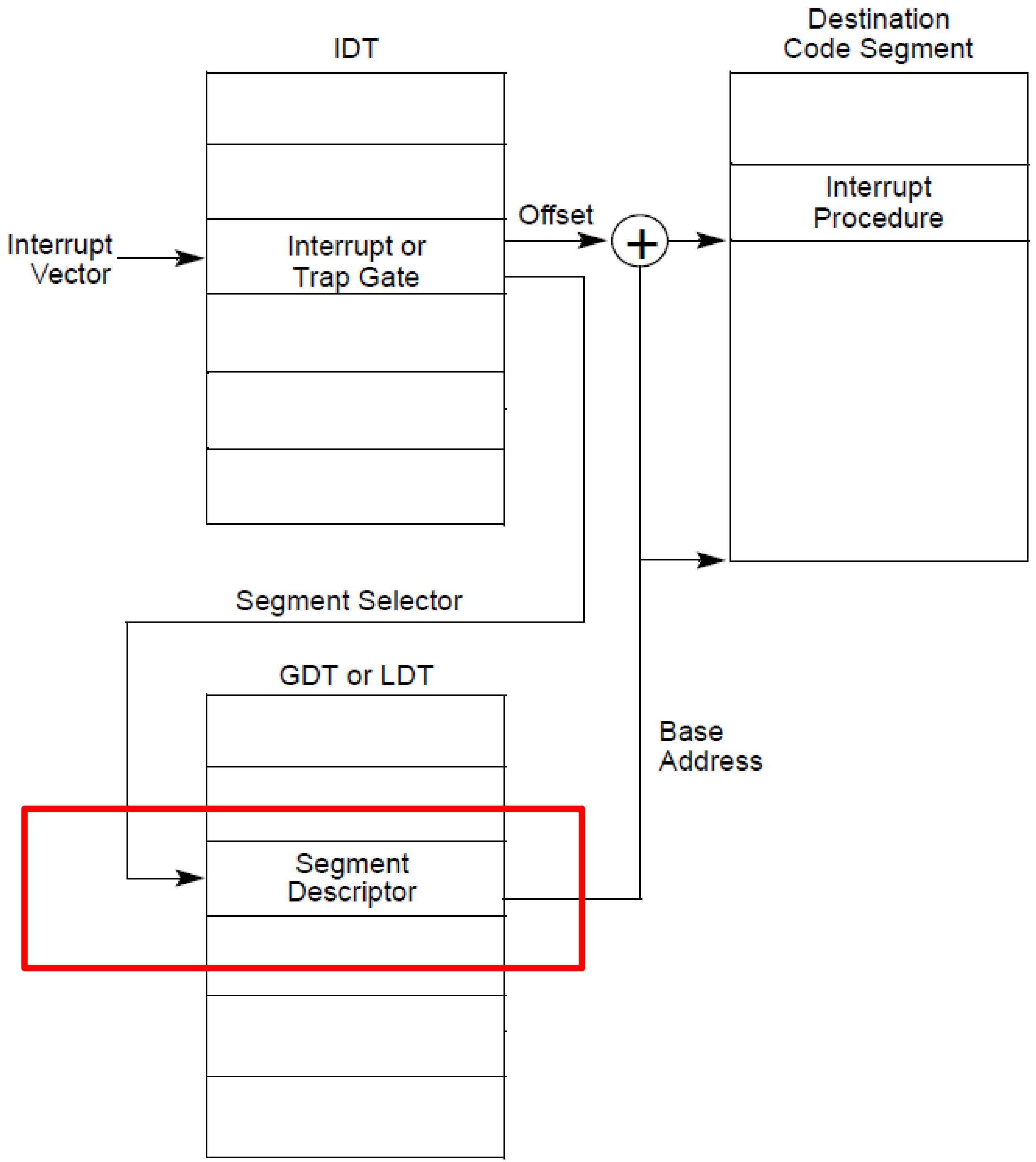
31 16 15 0

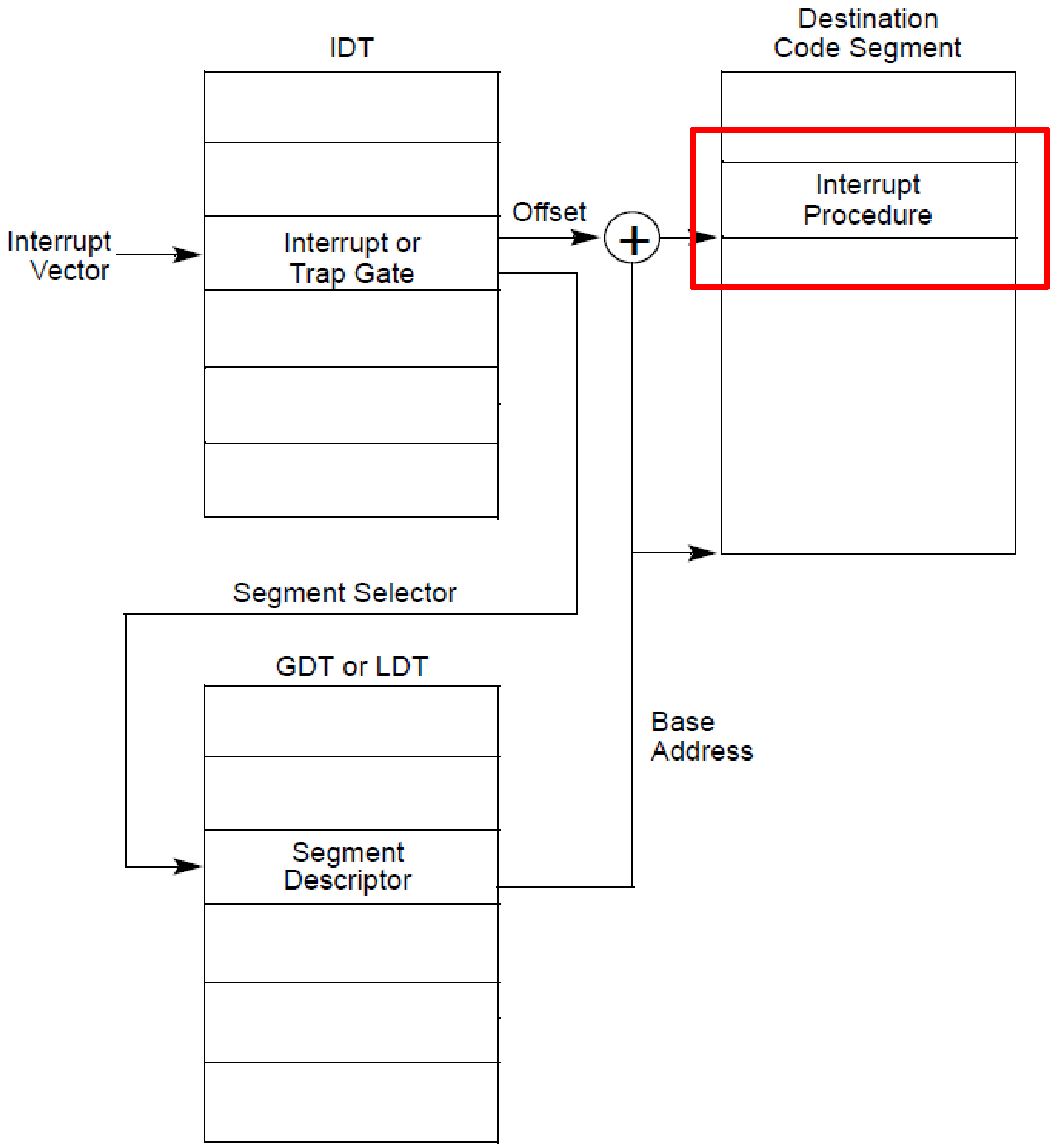
Segment Selector | Offset 15..0 | 0

Interrupt descriptor

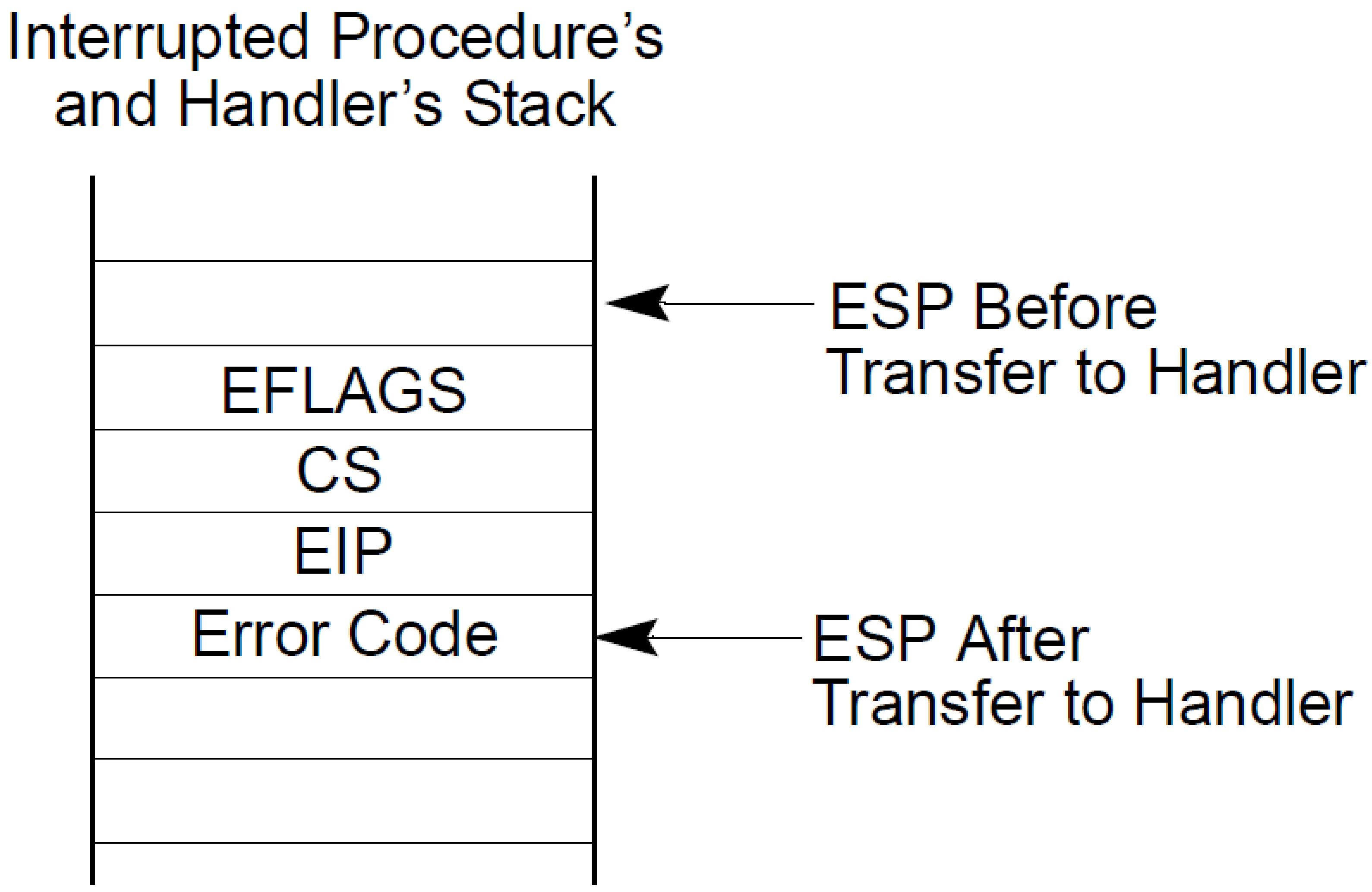



IDT
Destination
Code Segment
Interrupt
Vector
Interrupt or
Trap Gate
Offset
Interrupt
Procedure
Segment Selector
GDT or LDT
Base
Address
Segment
Descriptor


IDT
Destination
Code Segment
Interrupt
Vector
Interrupt or
Trap Gate
Offset
+
Interrupt
Procedure
Segment Selector
GDT or LDT
Base
Address
Segment
Descriptor


IDT
Destination
Code Segment
Interrupt
Vector
Interrupt or
Trap Gate
Offset
+
Interrupt
Procedure
Segment Selector
GDT or LDT
Base
Address
Segment
Descriptor

Stack Usage with No Privilege-Level Change



Interrupted Procedure's and Handler's Stack

EFLAGS

CS

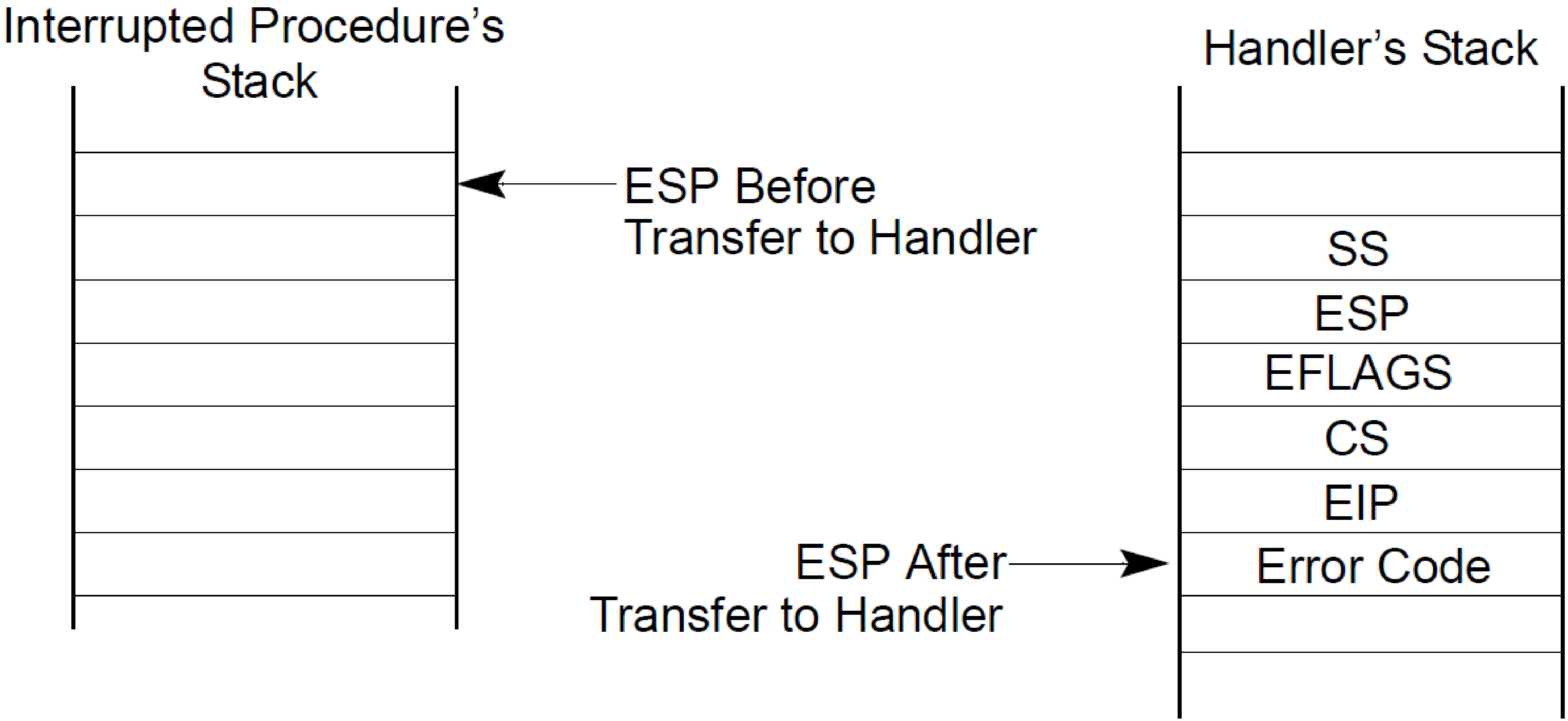
EIP

Error Code

ESP Before Transfer to Handler

ESP After Transfer to Handler

Stack Usage with Privilege-Level Change



Interrupted Procedure's Stack

Handler's Stack

ESP Before Transfer to Handler

SS

ESP

EFLAGS

CS

EIP

Error Code

ESP After Transfer to Handler

1. check that CPL <= DPL in the descriptor (but only if INT instruction).
2. save ESP and SS in a CPU-internal register (but only if target segment selector's PL < CPL).
3. load SS and ESP from TSS ("")
4. push user SS ("")
5. push user ESP ("")
6. push user EFLAGS
7. push user CS
8. push user EIP
9. clear some EFLAGS bits
10. set CS and EIP from IDT descriptor's segment selector and offset