Architettura x86 a 64 bit

Ovvero come l’architettura x86 a 32 bit è stata portata a 64 bit

Per cominciare

Contenuto

- Storia
- L’architettura a 32 bit
- L’architettura a 64 bit

Avvertenza

- Per abitudine ci si riferisce ai processori Intel e alla relativa terminologia, ma quanto esposto vale anche per i processori AMD (AMD ha iniziato con licenza Intel)
  - Fino al 486 vigeva la stessa identificazione dei modelli
  - Dal Pentium i numeri (non brevettabili) sono stati abbandonati e le sue società danno nomi diversi a CPU funzionalmente equivalenti (Pentium, Athlon, i7, Opteron)

Una nota

- Il titolo “Intel® 64 and IA-32 Architectures” compare nei manuali dal 2007
  - I manuali correnti sono sostanzialmente quelli che in precedenza erano intitolati “IA-32 Intel® Architecture”, con le dovute estensioni
    - Fino al 1999 erano intitolati “Intel Architecture”
- Chi in precedenza avesse fatto una ricerca con Google per “64 bit architecture” sarebbe approdato esclusivamente a pagine relative all’architettura **Itanium**

- Itanium è una famiglia completamente diversa. L’architettura venne definita da HP alla fine anni ’80. Accordo Intel-HP del 1994. Non ha avuto il successo sperato. Oggi non viene più valorizzata come prima
  - April 2010: Microsoft announces phase-out of support for Itanium
  - January 2011**:** Intel discontinues support for Itanium in its C/C++ and FORTRAN compilers[
  - March 2011, Oracle announced discontinuation of development on Itanium (nel 2012 dovrà riprendere per ingiunzione di tribunale)
  - 2012, SAP discontinues support for Business Objects on Itanium

- AMD è arrivata prima di Intel all’architettura a 64 bit (AMD64)

Architettura e Organizzazione

- **Architettura:** caratteristiche della macchina visibili al programmatore (ASM)
  - Formato dati numerici e testuali, numero e funzionalità dei registri di macchina, modalità di indirizzamento, repertorio di istruzioni, disponibilità di operazioni in virg. mobile, ecc.
  - (architettura ⇔ modello di programmazione)

- Il concetto di architettura è stato introdotto da IBM negli anni ’60.
  - Rende i programmi eseguibili su tutte le macchine che rispondono a una data architettura
    - I programmi scritti a suo tempo per un 386 girano su i recenti processori i3, i5, i7.

Architettura e Organizzazione

- **Organizzazione:** il modo in cui una data architettura è realizzata; le sottostanti parti strutturali; le relazioni tra le parti strutturali.
  (Implementazione dell'architettura)
  - Logica cablata o microprogrammata, presenza e organizzazione della cache, esecuzione in pipeline, metodo per la previsione dei salti, struttura dei bus, ecc.

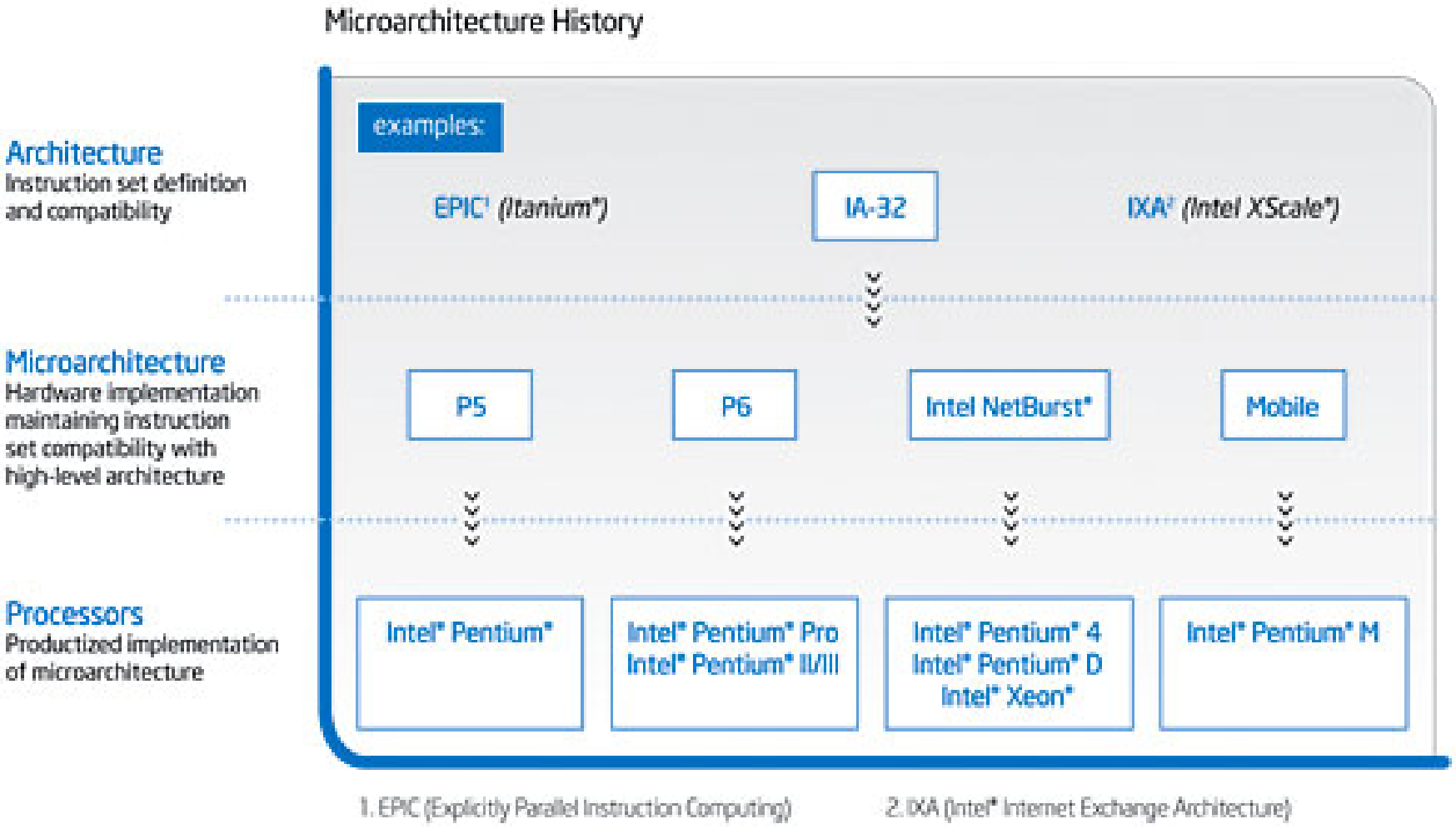
Architettura e Organizzazione

- **Organizzazione:** il modo in cui una data architettura è realizzata; le sottostanti parti strutturali; le relazioni tra le parti strutturali.
  (Implementazione dell'architettura)
  - Logica cablata o microprogrammata, presenza e organizzazione della cache, esecuzione in pipeline, metodo per la previsione dei salti, struttura dei bus, ecc.

Intel (e altri) usa il termine "microarchitettura" al posto di "organizzazione".

Trattando dell'organizzazione dei sistemi Intel, nel seguito verrà usato il termine "microarchitettura".

Architettura/microarchitettura



Sono proprio distinte?

- La distinzione tra Architettura e Microarchitettura non è netta
  - Di solito, ogni nuova microarchitettura aggiunge nuove caratteristiche che si riflettono sull'architettura (nuove istruzioni, nuovi aspetti di cui tener conto).
  - Di norma l'architettura risultante da una nuova microarchitettura incorpora la precedente (compatibilità all'indietro)

- Nel seguito ci interesseremo quasi esclusivamente dell'architettura, indipendentemente dalla sottostante organizzazione.

x86: Qualche dato del passato

	8086	80286	80386	80486	Pentium
anno di introduzione	1978	1982	1985	1989	1993
tecnologia	HMOS	HMOS	CMOS	CMOS	Bi-CMOS
(dimensione)	(3 μm)	(2 μm)	(1,5 μm)	(1 μm)	(0,8 μm)
n° transistor	29.000	134.000	275.000	1.200.000	3.100.000
n° piedini	40	68	132	168	273
freq. int/est	8/8	12/12	33/33	66/33	200/66
freq. introduzione	5	8	16	25	60
registri (bit)	16	16	32	32	32
bus dati (bit)	16	16	32	32	64
indirizzi (bit)	20	24	32	32	32
mem virtuale	no	seg.	seg.+pag.	seg.+pag.	seg.+pag.
spazio ind reale/virt.	1MB/-	16MB/1GB	4GB/64TB	4GB/64TB	4GB/64TB
aritm floating	no	no	no	si	si
cache (on chip)	no	no	no	si	si
MIPS (alla f intro)	0,33	1,2	3,6	20	100

Il 386 introduce l'architettura a 32 bit

Dal primo all'ultimo (Intel)

Data di introduzione	Nome del chip	N. di transistori (/1000)	Tecnologia (μm)	Frequenza (MHz)
Novembre 1971	4004	2,3	10	0,108
Giugno 1978	8086	29	3	5
Marzo 1993	Pentium	3.100	0,8	60
Novembre 1995	PentiumPro	5.500	0,6	150
Novembre 2000	Pentium 4	42.000	0,18	1.400
Marzo 2003	Pentium M	77.000	0,13	1.300
Marzo 2006	Core Duo	152.000	0,065	1.060
Luglio 2006	Mobile Core 2 Duo	167.000	0,065	1.060
Gennaio 2008	Core 2 Quad	820.000	0,045	2.500
Giugno 2008	Atom	47.000	0,045	1.600
Luglio 2010	Six Core i7-970	1.170.000	0,032	3.200

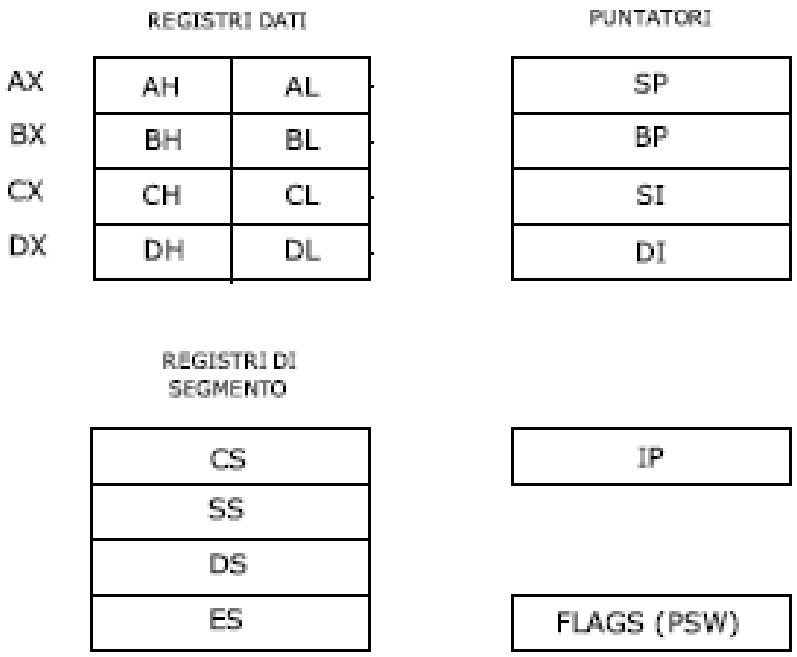
Consumo: 4W

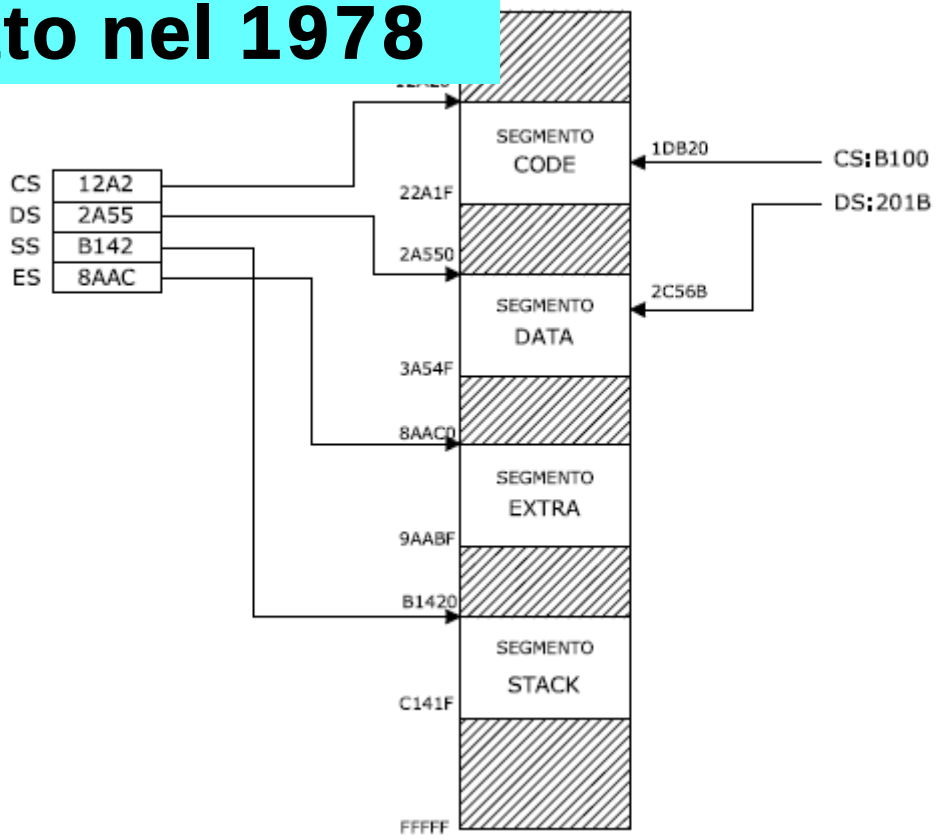
6 core, 12 thread
Consumo: 130 W

Modello di programmazione (8086)

- Registri a 16 bit (con funzionalità distinte)
- Memoria segmentata

Tutto è cominciato nel 1978





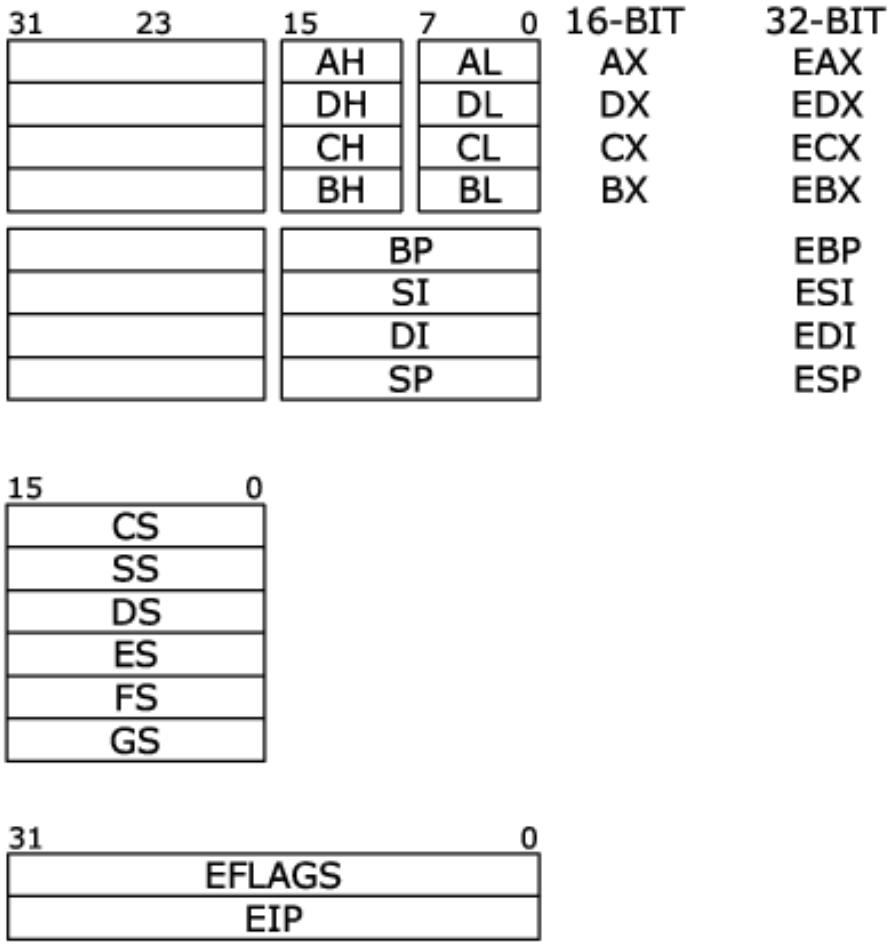
Architettura a 32 bit

- Introdotta con il processore 80386 (386), nel 1985
- Registri a 32 bit per dati e indirizzi
- I 16 bit meno significativi dei registri mantengono le stesse funzionalità dei registri a 16 bit delle precedenti CPU (compatibilità all'indietro)
- Bus dati e indirizzi a 32 bit per un indirizzamento fino a 4 GB di memoria fisica
- Offset codificato nelle istruzioni fino a 32 bit
- Modello di memoria segmentata
- Memoria virtuale paginata (a valle della segmentazione), pagine di 4K
- Modo di funzionamento "virtual-8086"

- Intel la denomina ***Architettura IA-32,*** AMD la denomina ***AMD32***
  - Useremo la sigla **x32** (che sia Intel o AMD o altro)
- E' la base dell'architettura a 64 bit (che Intel denomina ***Architettura Intel 64,*** AMD la denomina ***AMD64***), che di fatto rappresenta una sua estensione
  - Per questa useremo la sigla **x64**

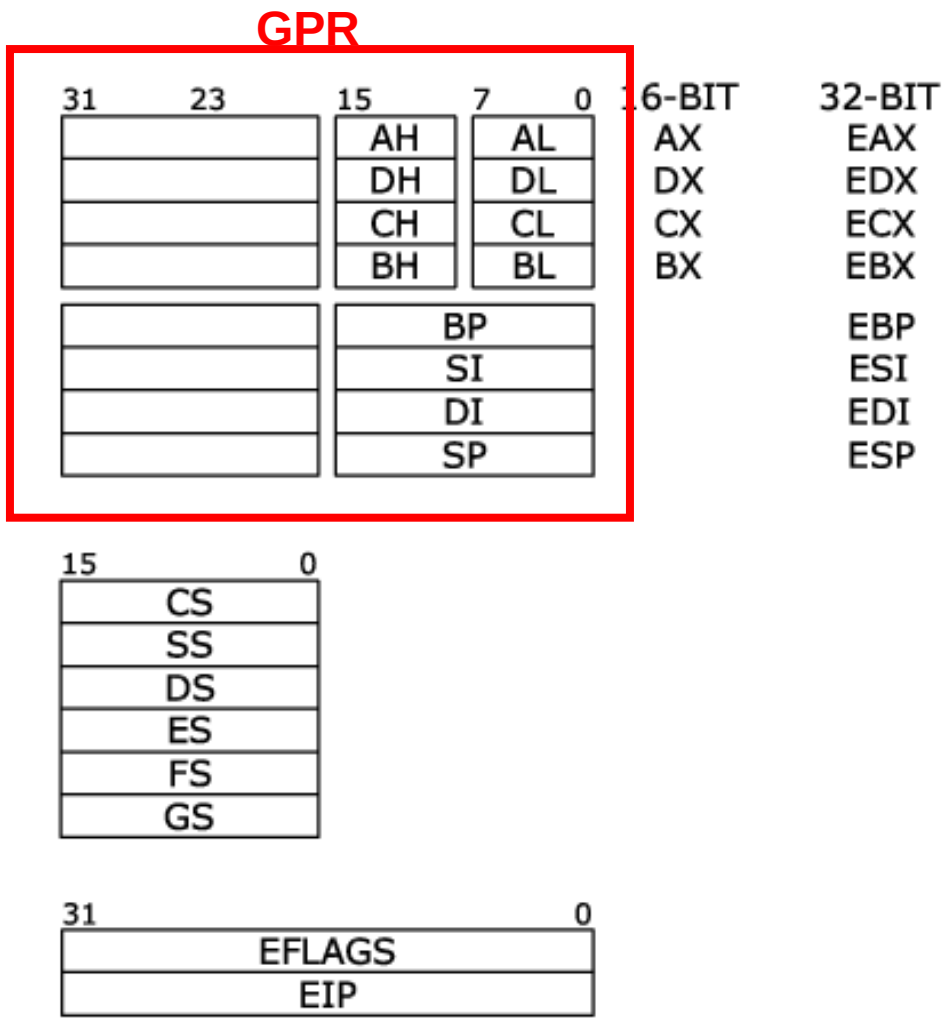
Modello (base) di programmazione 386

- Comune a tutti i modelli successivi, a parte minori differenze
- Nel corso degli anni al modello di base si sono aggiunte svariate estensioni (x87 integrato, MMX, SSE)

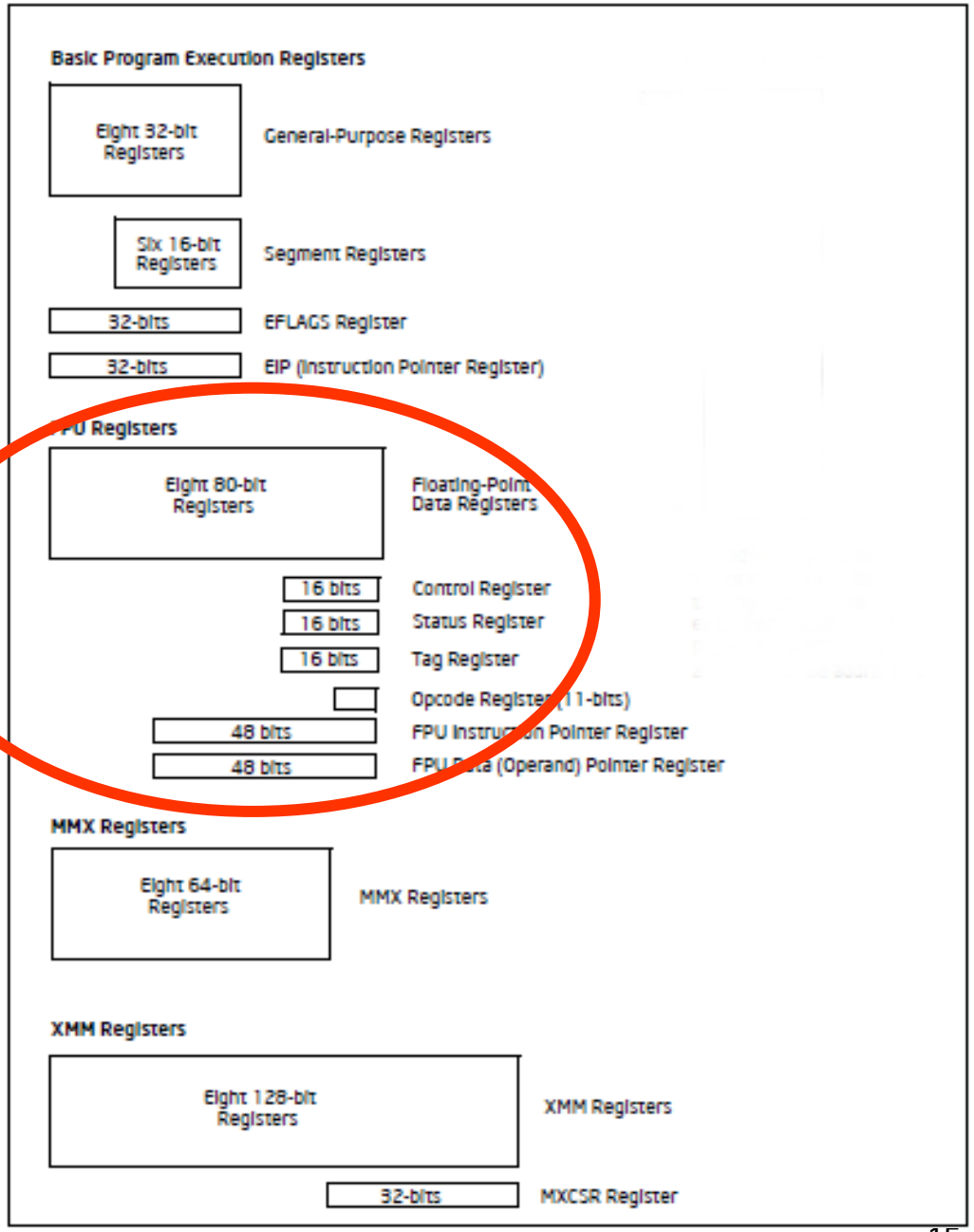


Modello (base) di programmazione x32

- I GPR a 32 bit sono tra loro completamente intercambiabili (non esistono più specializzazioni o limitazioni al loro uso)
- Hanno mantenuto le vecchie denominazioni per compatibilità nella programmazione

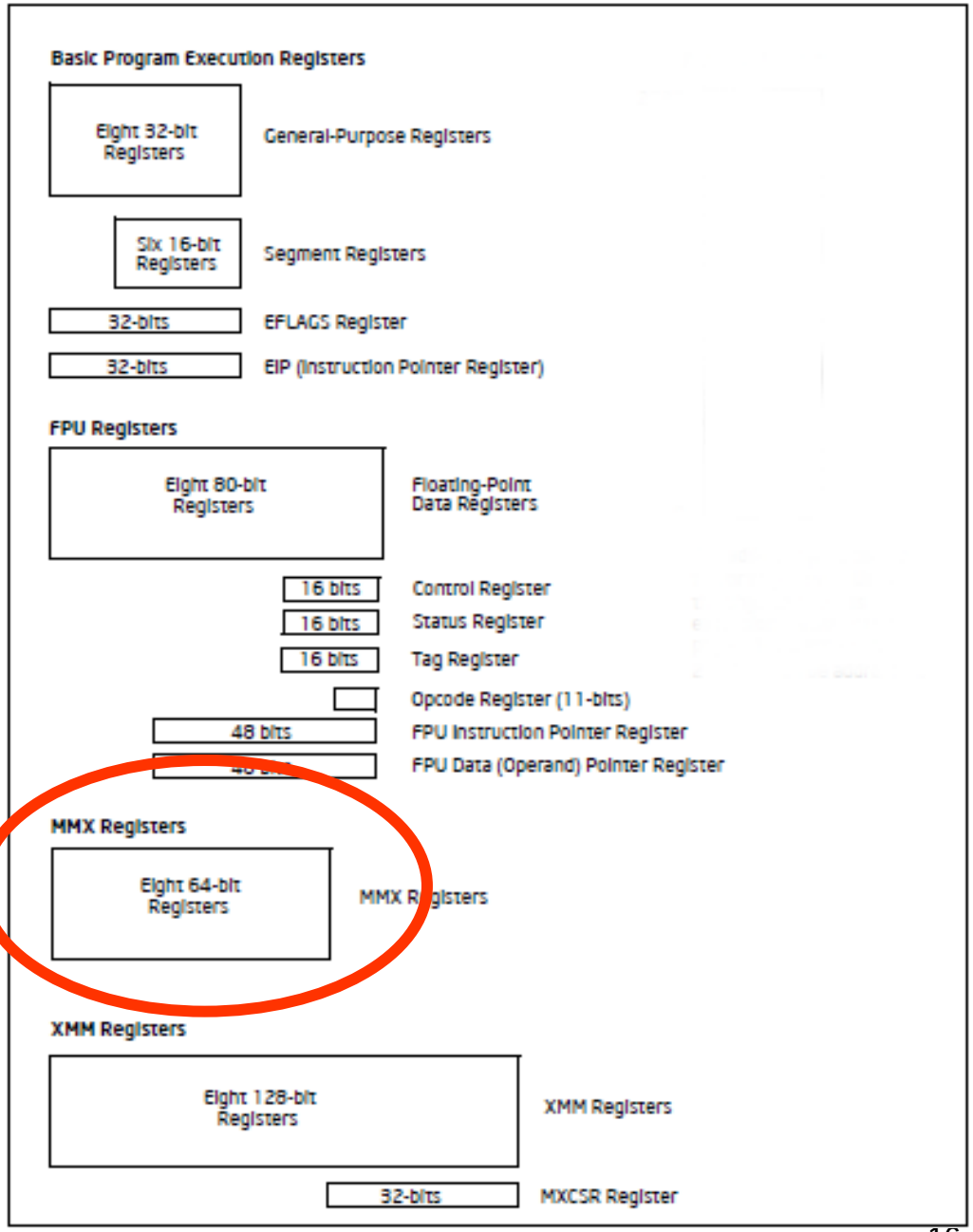


x32 Estensioni al modello base: FPU x87 (integrata dal 486)



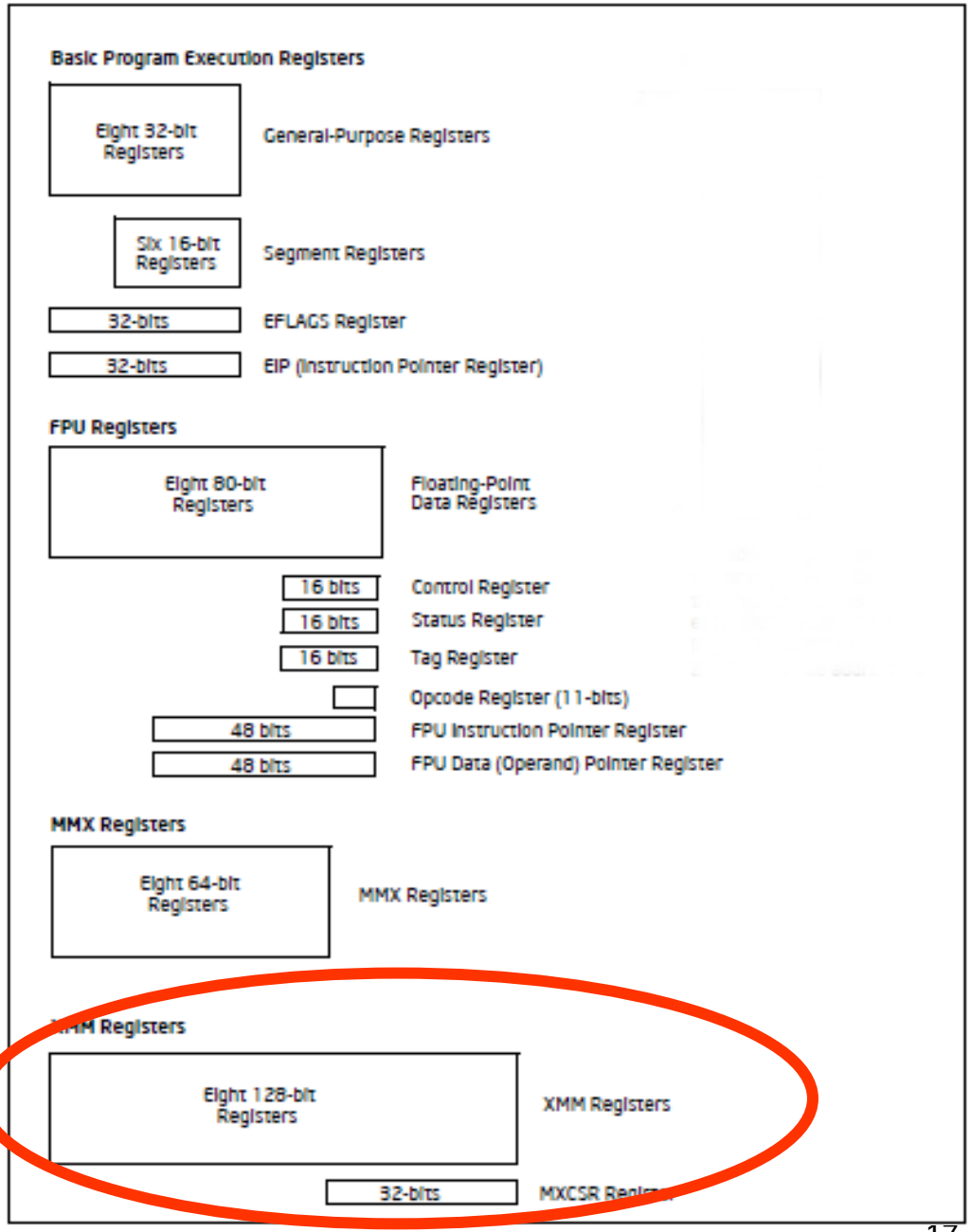
x32
Estensioni al modello base: Multimedia Extension
(dal Pentium MMX)

Istruzioni di tipo SIMD (*Single Instruction Multiple Data*).
Usa gli stessi registri della FPU: l'uso di MMX e FPU non può essere contemporaneo)

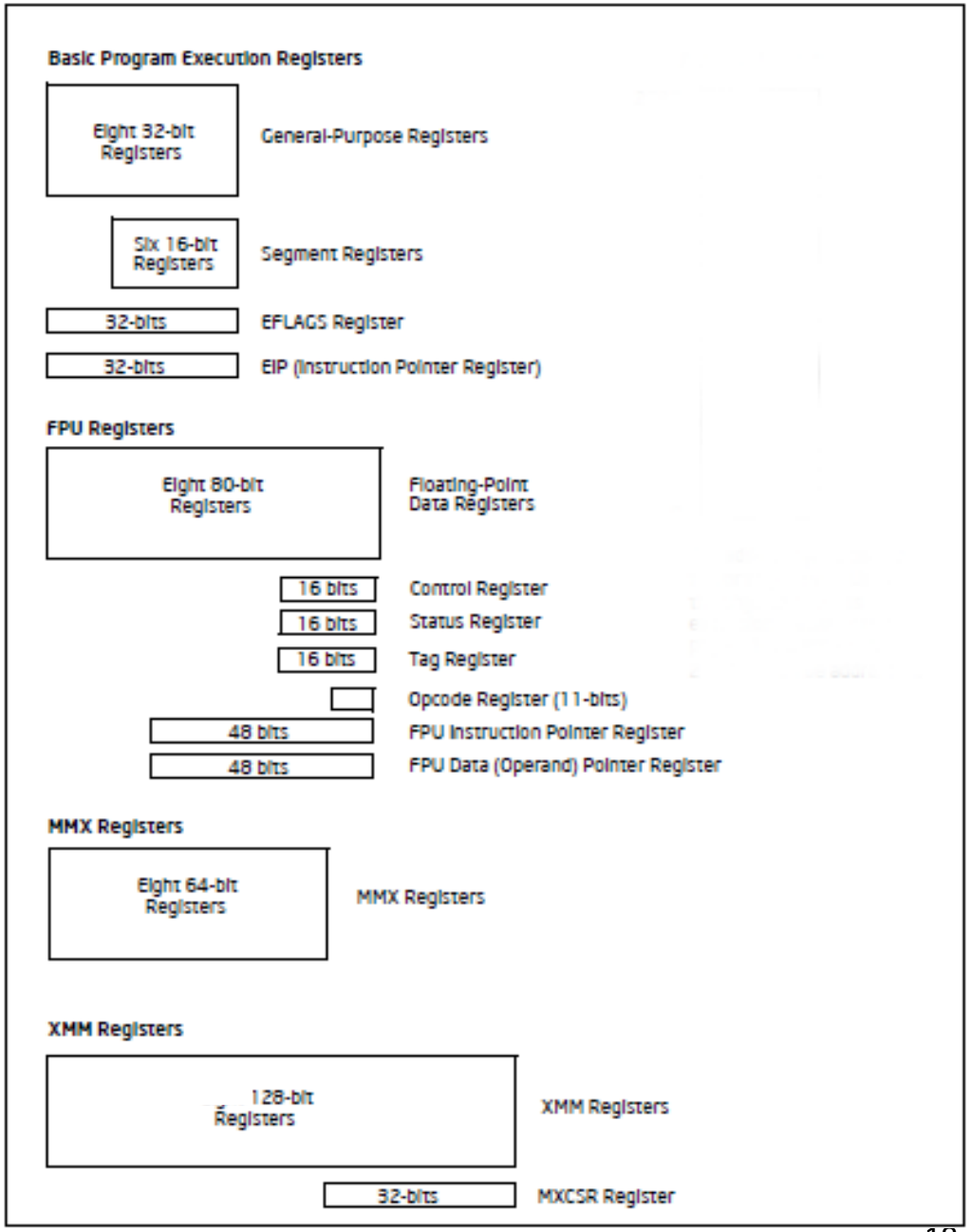


x32
Estensioni al modello base: Streaming SIMD Extension (SSE)
(dal Pentium III)

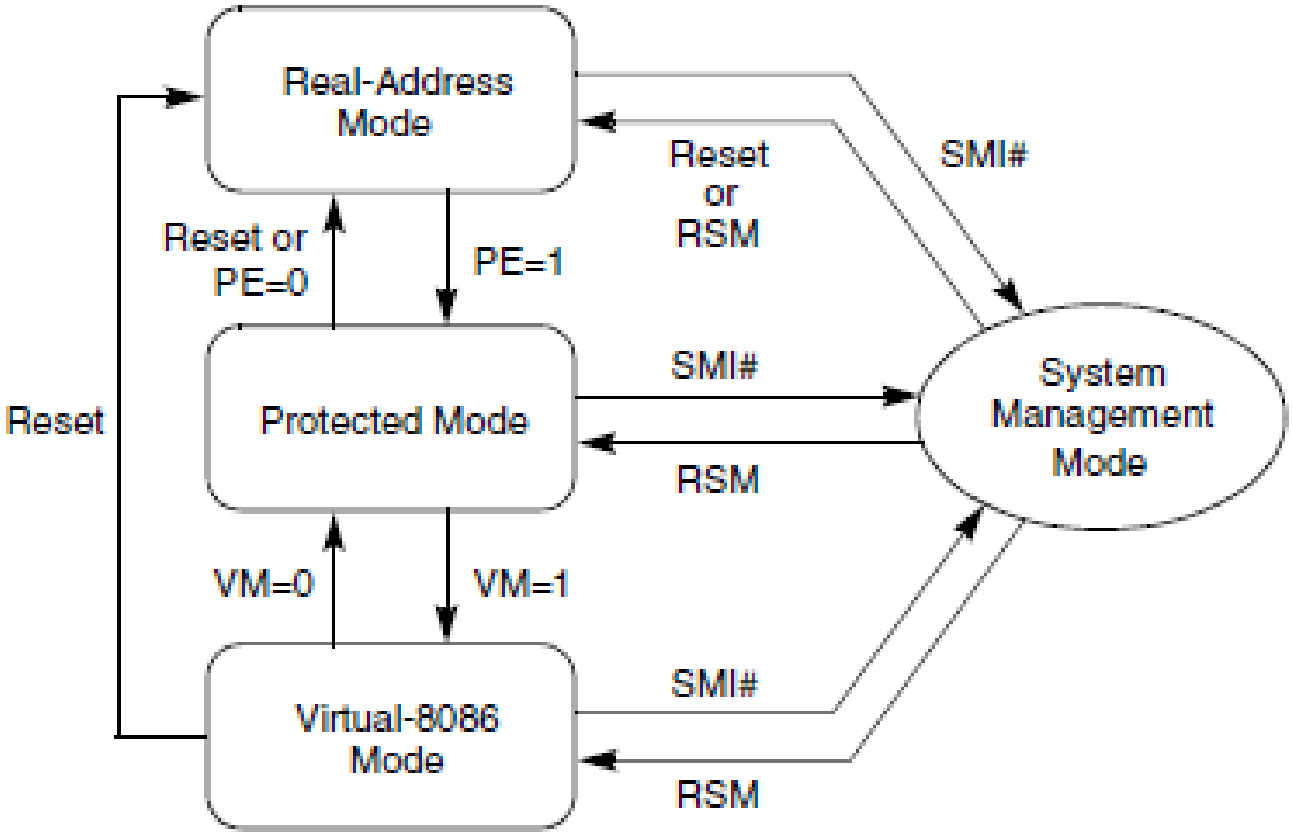
Istruzioni di tipo SIMD su registri separati a 128 bit
(non richiedono la disabilitazione di FPU)



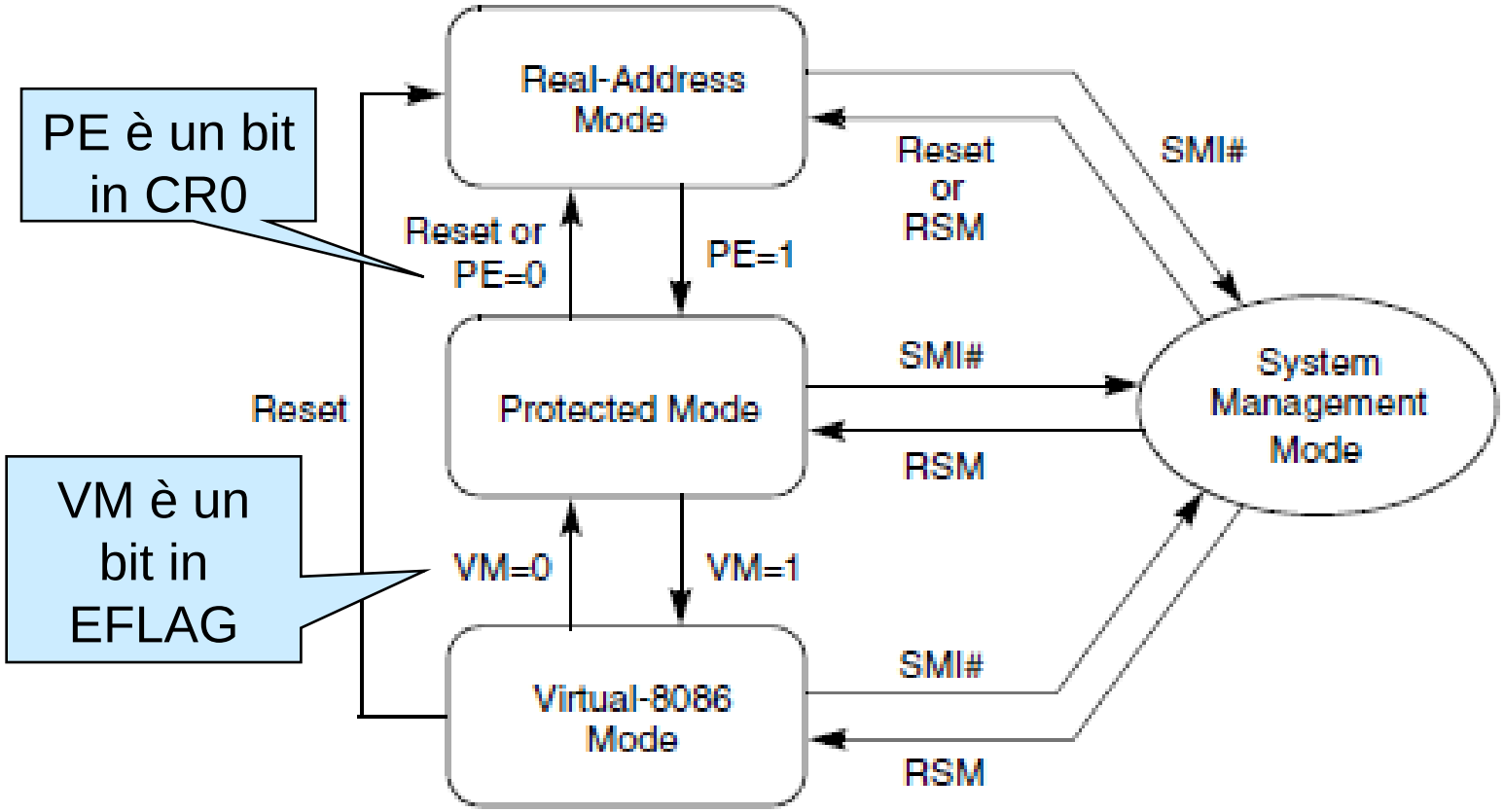
Con i modelli attuali si è arrivati alla versione SSE 4.2



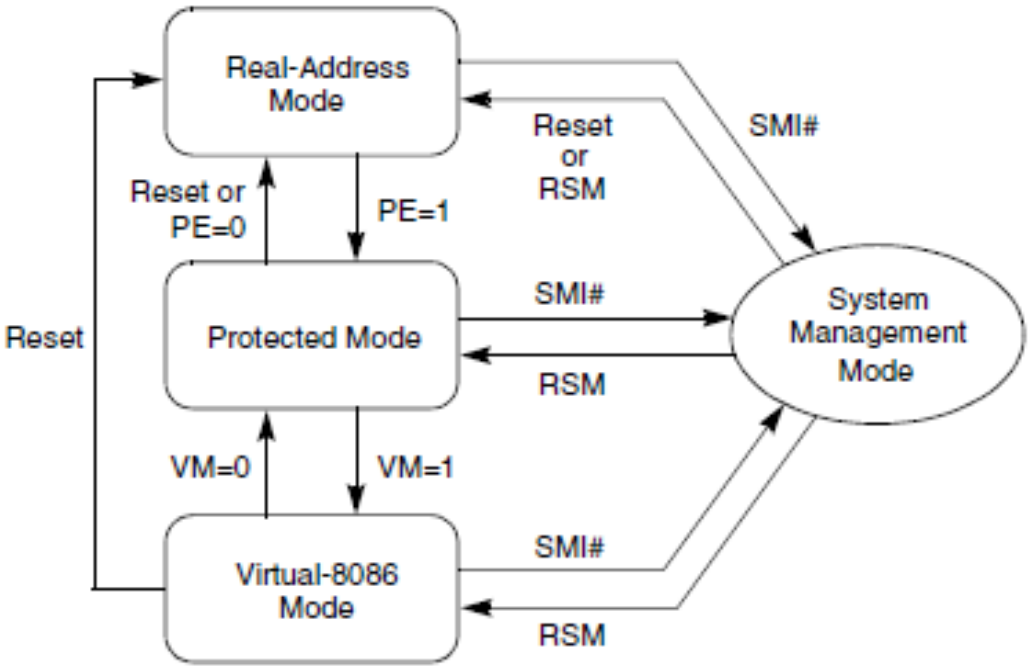
Modi di esecuzione x32



Modi di esecuzione x32



Modi di esecuzione x32

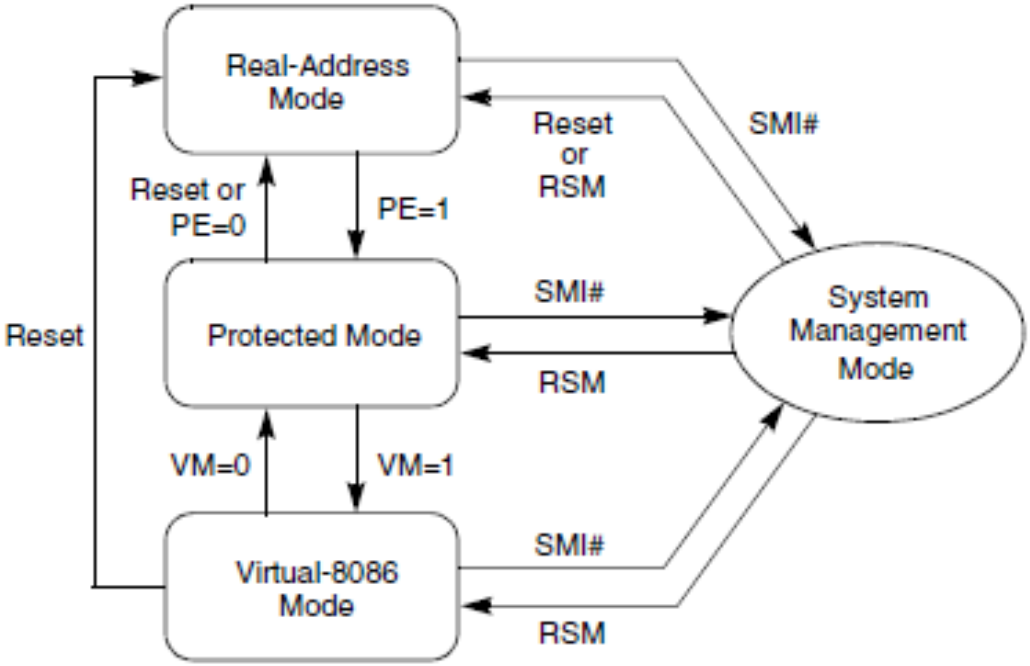


Real-address mode:

Come un 8086

All'accensione qualunque processore viene messo in questo stato

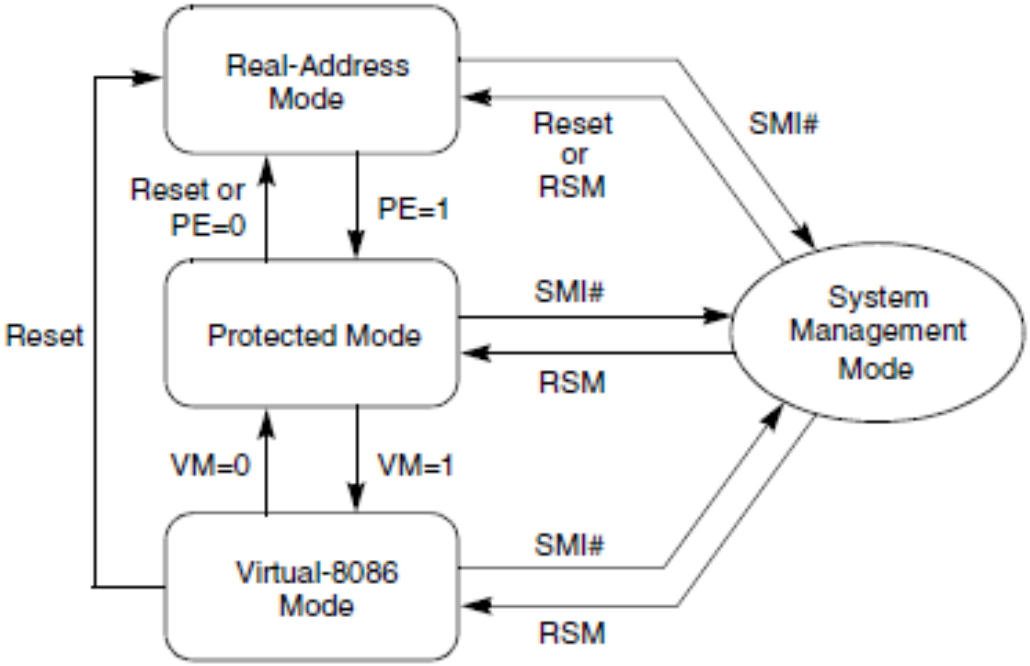
Modi di esecuzione x32



Protected mode: modalità standard (“nativa”)
Implementa un sistema multitasking (protetto) e la memoria virtuale

Virtual-8086 mode: sottomodo del protected. Esegue come un 8086 (ma entro un task protetto)

Modi di esecuzione x32



System management mode (SMM)

Fornisce al sistema operativo un meccanismo “trasparente” per l'implementazione di funzioni specifiche.

Presenta verso l’esterno il segnale SMM con il quale si può indirizzare uno spazio di memoria completamente distinto.

Passaggio da 32 a 64 bit - Problemi

- Estensione del parallelismo (registri)
- Le istruzioni devono specificare se il dato trasferito è a 8, 16, 32 ovvero 64 bit
- Mantenere lo stesso modello di memoria ?

IMPERATIVO: **Mantenere la compatibilità con i programmi scritti per macchine precedenti a 16/32 bit**

- Similmente a quanto fatto nel passaggio da 16 a 32 bit

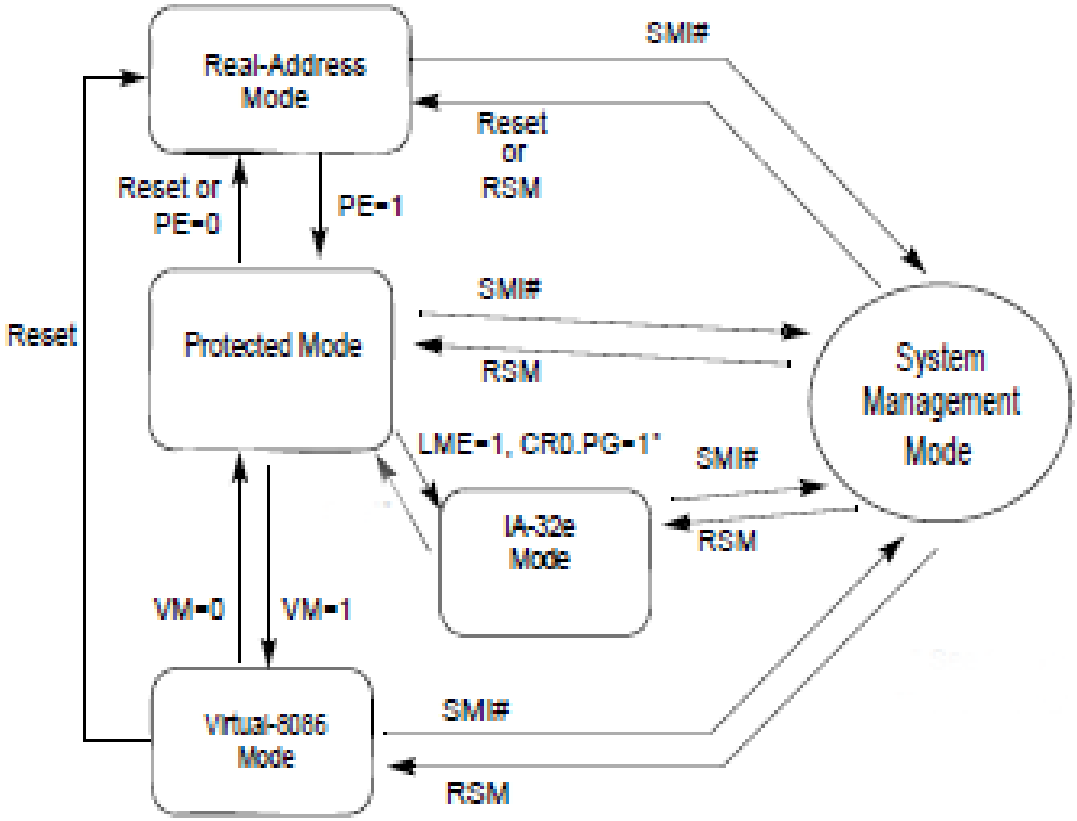
Passaggio da 32 a 64 bit - Soluzioni

- Introdurre una nuova modalità operativa (*), con due sottomodalità
  - Compatibility mode
  - 64-bit mode
- Mantenere invariato il formato delle istruzioni (scostamenti a 32 bit), prevedendo un nuovo “prefisso” per trattare registri a 64 bit
- Passare al modello di memoria lineare nella sottomodalita 64 bit

(*) Chiamata “IA-32e” da Intel e “Long” da AMD

Passaggio da 32 a 64 bit - Soluzioni

- Introdurre una nuova modalità operativa (x32e, ovvero x32 estesa), con due sottomodalità
- Mantenere invariato il formato delle istruzioni (scostamenti a 32 bit), prevedendo un nuovo “prefisso” per trattare registri a 64 bit
- Passare al modello di memoria lineare nella sottomodalita 64 bit



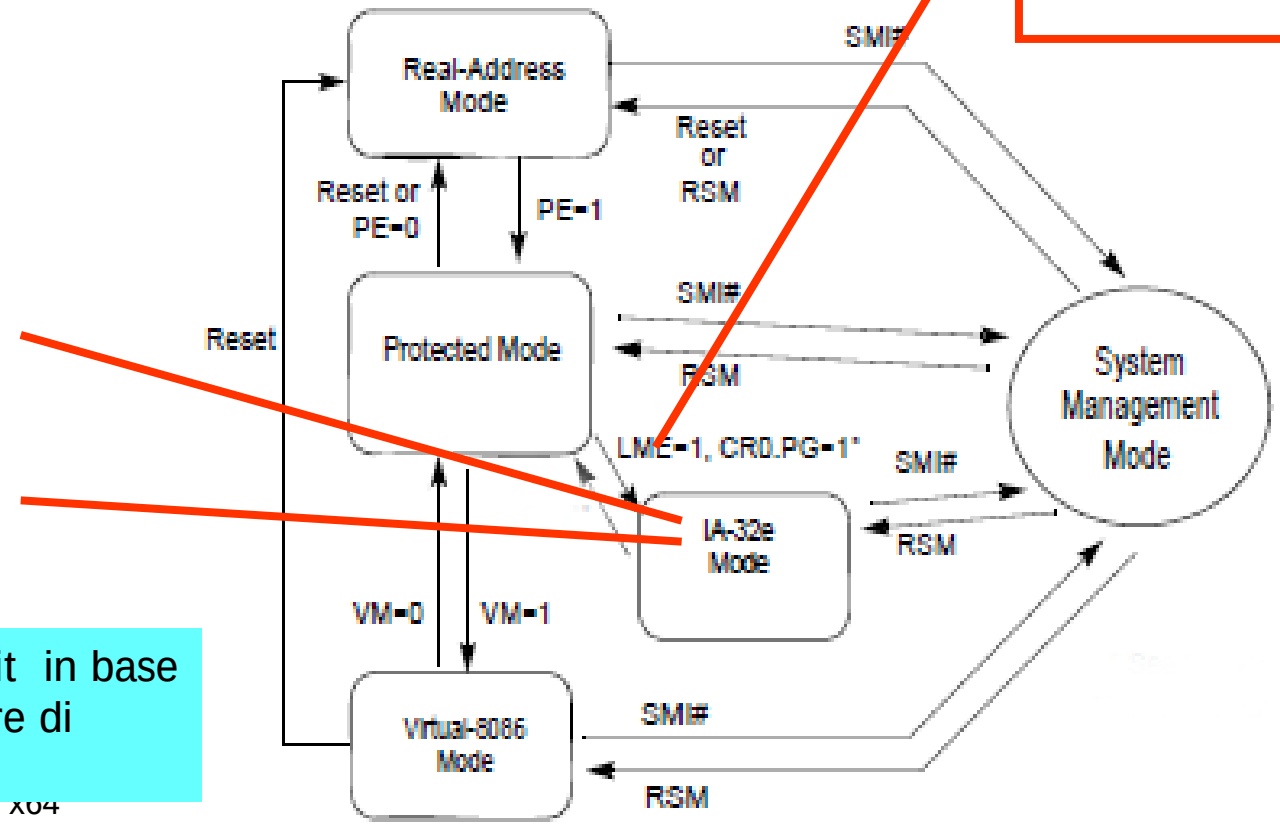
Passaggio da 32 a 64 bit - Soluzioni

- Introdurre una nuova modalità operativa (x32e, ovvero x32 estesa), con due sottomodalità
  - Mantenere invariato il formato delle istruzioni (scostamenti a 32 bit), prevedendo un nuovo "prefisso" per trattare registri a 64 bit
  - Passare al modello di memoria lineare nella sottomodalita 64 bit

LME è un bit uno speciale registro (EFER)

Compatibilty mode

64-bit mode



Compatibility o 64 bit in base al bit L del descrittore di segmento in CS

Modi di esecuzione a 64 bit

Il modo x32e, che ha due sotto modi:

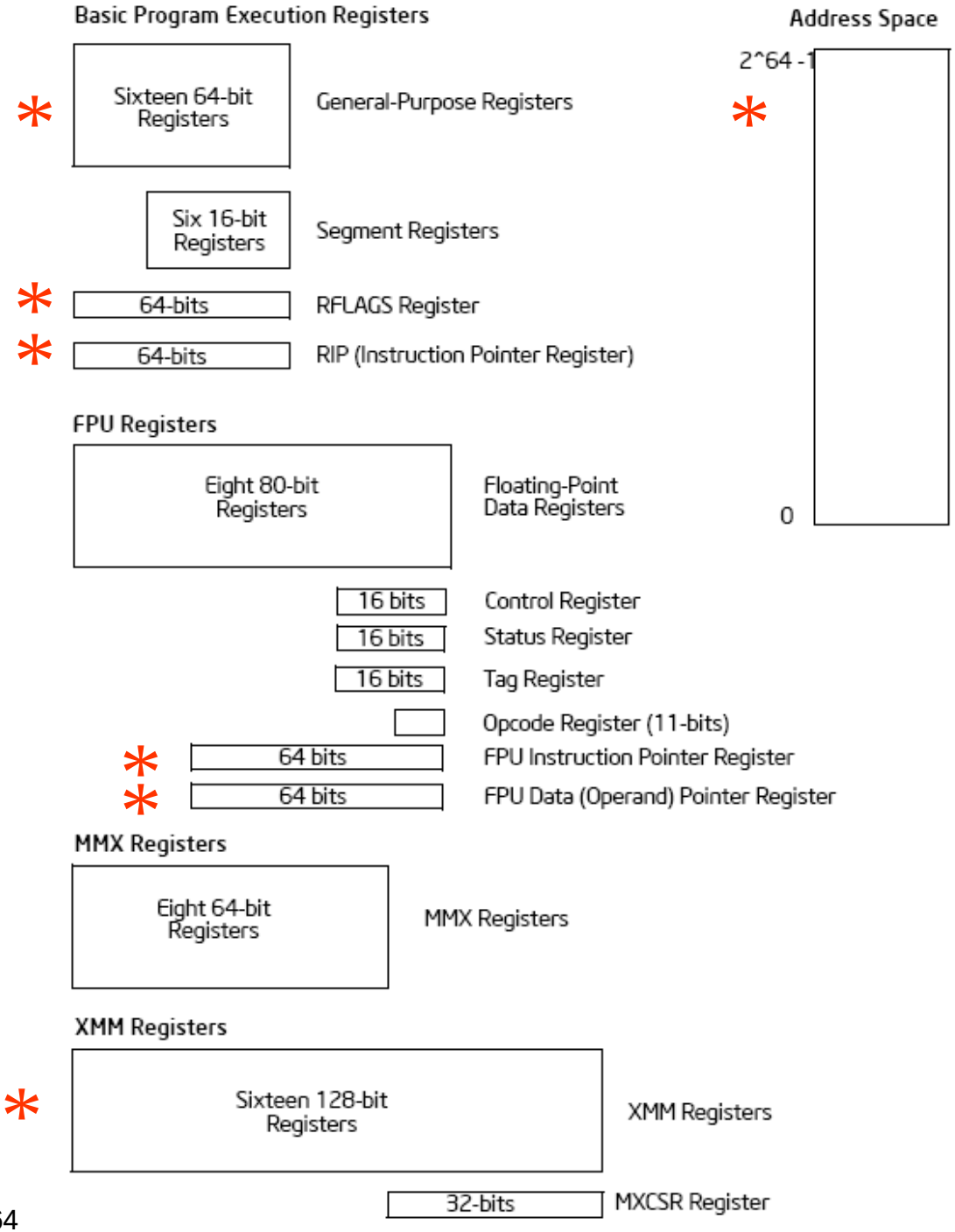
- **64-bit mode (sub-mode of x32e mode)** — Presuppone un sistema operativo a 64 bit, sotto cui eseguono applicazioni a 64 bit che vedono uno spazio lineare ampio 2^{64} bit.
  - Il numero di registri di uso generale (ora a 64 bit) passa da 8 a 16.
- **Compatibility mode (sub-mode di x32e mode)** — Consente alle precedenti applicazioni a 32 o 16 bit di girare sotto un sistema operativo a 64 bit. Senza che ci sia bisogno di ricompilarle
  - Le applicazioni vedono solo i primi 4 GB dello spazio lineare; L'indirizzamento può essere a 16 o 32 bit.

IMPERATIVO rispettato !

64-Bit Mode Execution Environment

(Estensione del parallelismo a 64 bit)

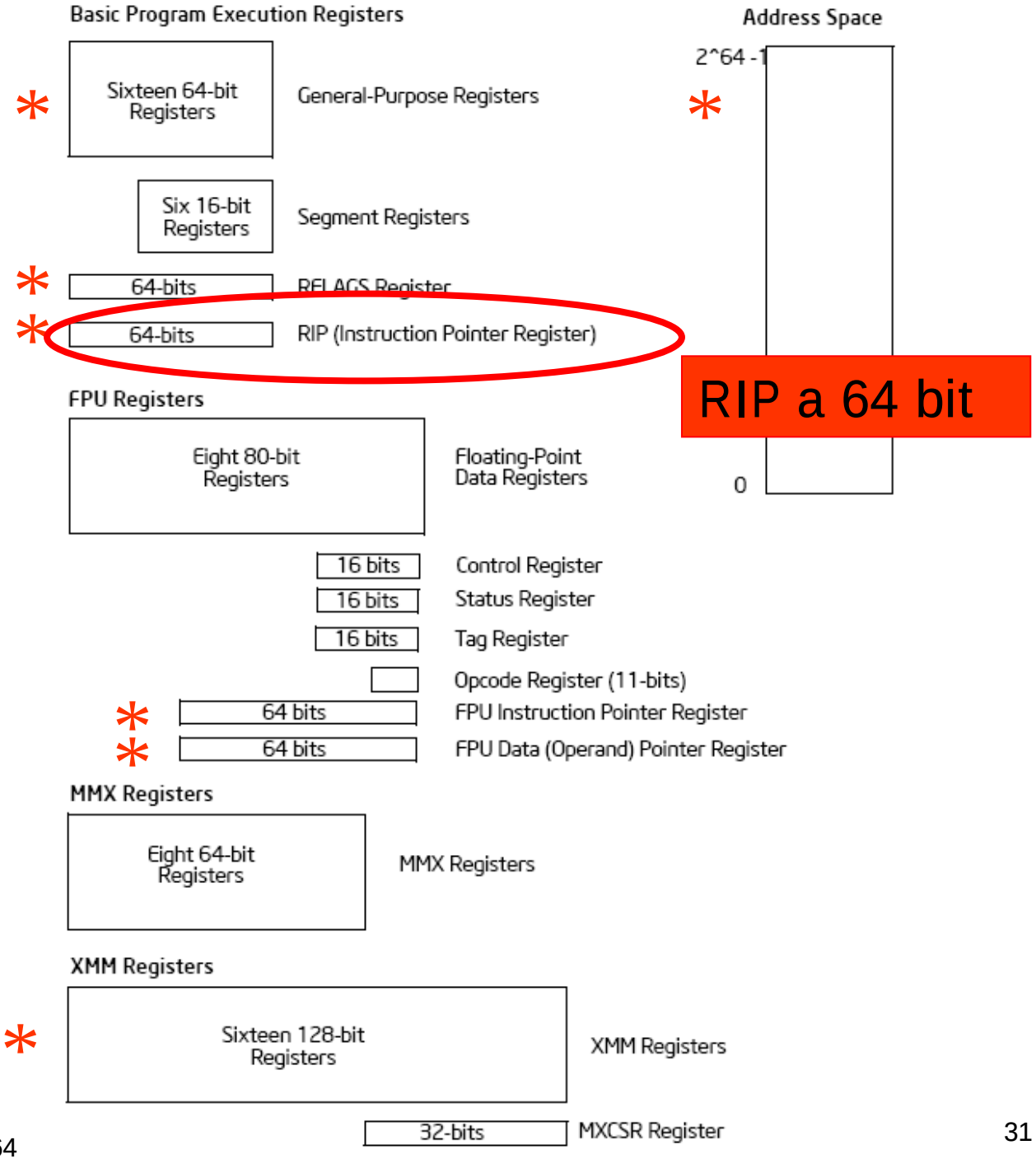
* variazioni



64-Bit Mode
Execution Environment

(Estensione del parallelismo a 64 bit)

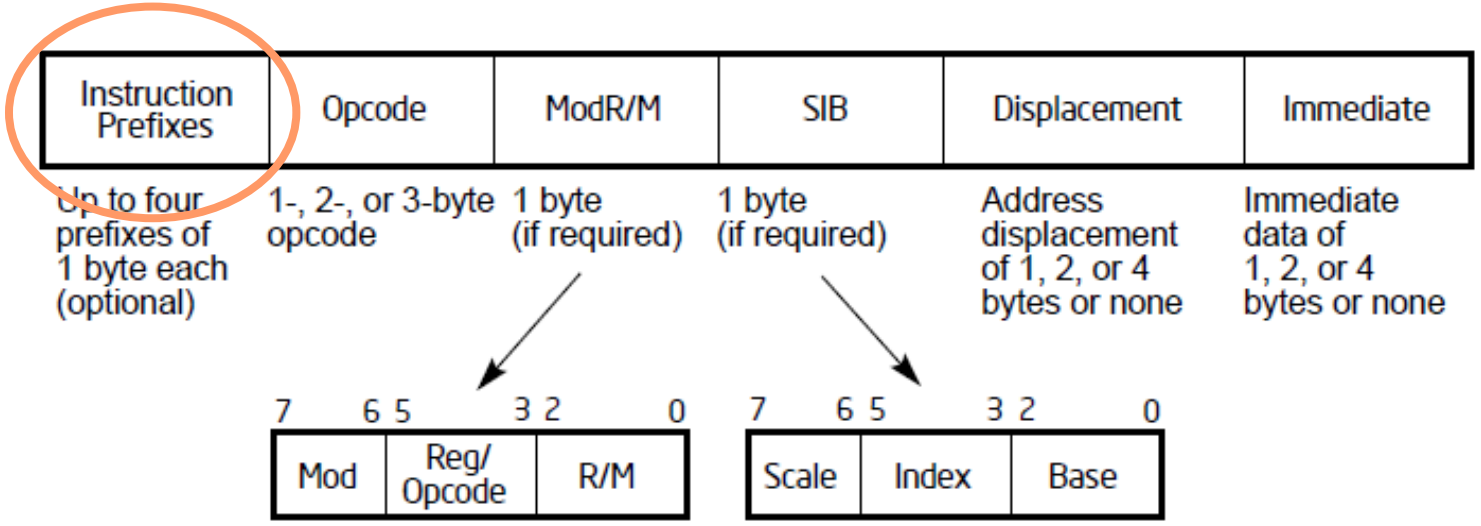
* variazioni



Passaggio da 32 a 64 bit - Soluzioni

- Introdurre una nuova modalità operativa (x32e), con due sottomodalità
- Mantenere invariato il formato delle istruzioni (scostamenti a 32 bit), prevedendo un nuovo “prefisso” per trattare registri a 64 bit
- Passare al modello di memoria lineare nella sottomodalita 64 bit

Formato istruzioni x32 (da tipica macchina CISC)



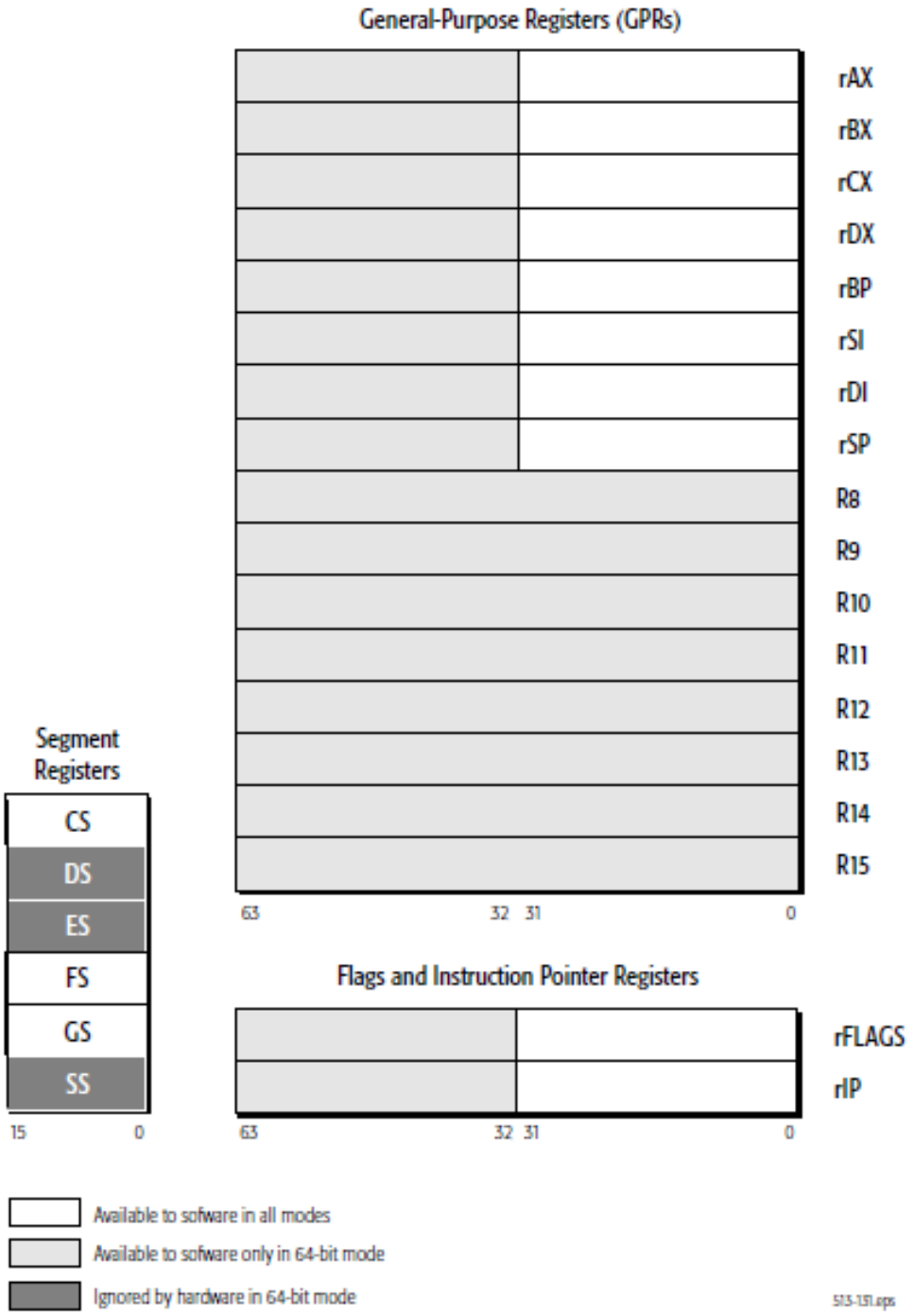
Passaggio da 32 a 64 bit - Soluzioni

- Introdurre una nuova modalità operativa (x32e), con due sottomodalità
- Mantenere invariato il formato delle istruzioni (scostamenti a 32 bit), prevedendo un nuovo “prefisso” per trattare registri a 64 bit
- Passare al modello di memoria lineare nella sottomodalita 64 bit

Legacy Prefixes	REX Prefix	Opcode	ModR/M	SIB	Displacement	Immediate
Grp 1, Grp 2, Grp 3, Grp 4 (optional)	(optional)	1-, 2-, or 3-byte opcode	1 byte (If required)	1 byte (If required)	Address displacement of 1, 2, or 4 bytes	Immediate data of 1, 2, or 4 bytes or none

Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8L - R15L
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

Registri


General-Purpose Registers (GPRs)
rAX
rBX
rCX
rDX
rBP
rSI
rDI
rSP
R8
R9
R10
R11
R12
R13
R14
R15
63
32
31
0
Segment Registers
CS
DS
ES
FS
GS
SS
15
0
Flags and Instruction Pointer Registers
rFLAGS
rIP
63
32
31
0
Available to software in all modes
Available to software only in 64-bit mode
Ignored by hardware in 64-bit mode

Passaggio da 32 a 64 bit - Soluzioni

- Introdurre una nuova modalità operativa (x32e), con due sottomodalità
- Mantenere invariato il formato delle istruzioni (scostamenti a 32 bit), prevedendo un nuovo “prefisso” per trattare registri a 64 bit
- Passare al modello di memoria lineare nella sottomodalita 64 bit

Legacy Prefixes	REX Prefix	Opcode	ModR/M	SIB	Displacement	Immediate
Grp 1, Grp 2, Grp 3, Grp 4 (optional)	(optional)	1-, 2-, or 3-byte opcode	1 byte (if required)	1 byte (if required)	Address displacement of 1, 2, or 4 bytes	Immediate data of 1, 2, or 4 bytes or none

Register Type	Without REX	With REX
Byte Registers	AL, BL, CL, DL, AH, BH, CH, DH	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8L - R15L
Word Registers	AX, BX, CX, DX, DI, SI, BP, SP	AX, BX, CX, DX, DI, SI, BP, SP, R8W - R15W
Doubleword Registers	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D - R15D
Quadword Registers	N.A.	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8 - R15

NB:
i displacement restano a 32 bit al massimo

Passaggio da 32 a 64 bit - Soluzioni

- Introdurre una nuova modalità operativa (x32e), con due sottomodalità
- Mantenere invariato il formato delle istruzioni (scostamenti a 32 bit), prevedendo un nuovo “prefisso” per trattare registri a 64 bit
- Passare al modello di memoria lineare nella sottomodalita 64 bit

- Questa è una variazione molto rilevante
- Vuol dire abbandonare il modello di memoria dell’architettura x86
  - Sotto c’è ancora il modello segmentato che però viene “anestetizzato”

Occorre fare una digressione sulla memoria

Memoria (definizioni)

- **Memoria Fisica**: a ogni byte è associato un indirizzo fisico

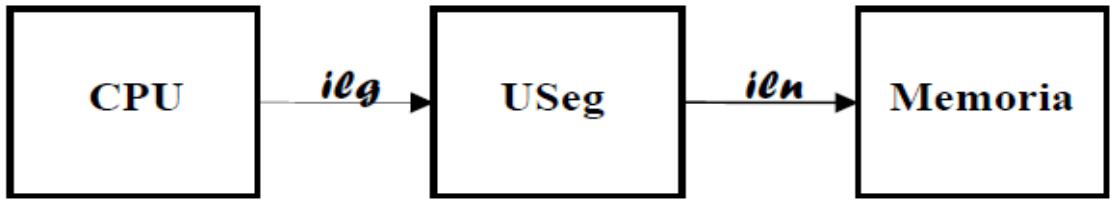
- **Indirizzo fisico**: è necessariamente lineare
  - va da 0 a M-1 (essendo M lo spazio di memoria indirizzabile)

- **Indirizzo logico (**ovvero **virtuale)**: indirizzo generato dalla (parte interna della) CPU

Memoria (definizioni)

L'indirizzo logico risponde a uno di questi 2 modelli:

- **Modello lineare**: L'indirizzo logico è lineare {0, M-1}
  - si riporta direttamente sull'indirizzo fisico
- **Modello segmentato**: L'indirizzo logico ha due componenti (**SR : OFFSET**)
  - La memoria è vista come formata da spazi di indirizzi indipendenti chiamati segmenti da allocare nella memoria fisica
  - Comporta la traduzione da indirizzo logico a indirizzo lineare (attraverso una Unità di segmentazione)



La segmentazione è l'aspetto più caratteristico dell'architettura x86

Memoria (definizioni)

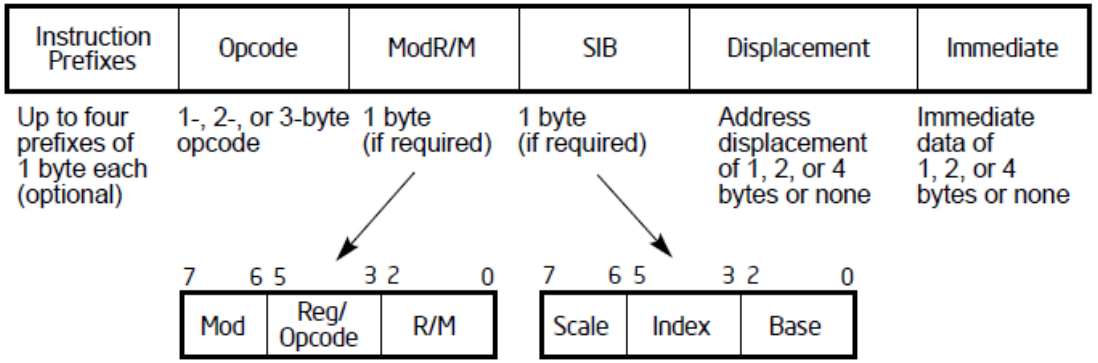
- **Offset** calcolato come
  - **`Base + (Scale x Index) + Displacement`**
    - **`Base`**: valore contenuto in qualunque GPR
    - **`Scale`**: un valore tra 1, 2, 4, or 8
    - **`Index`**: valore in complemento a 2 (+ o -) in qualunque GPR
    - **`Displacement`**: un valore in complemento a 2 su 8-bit, 16-bit, or 32-bit codificato nell'istruzione

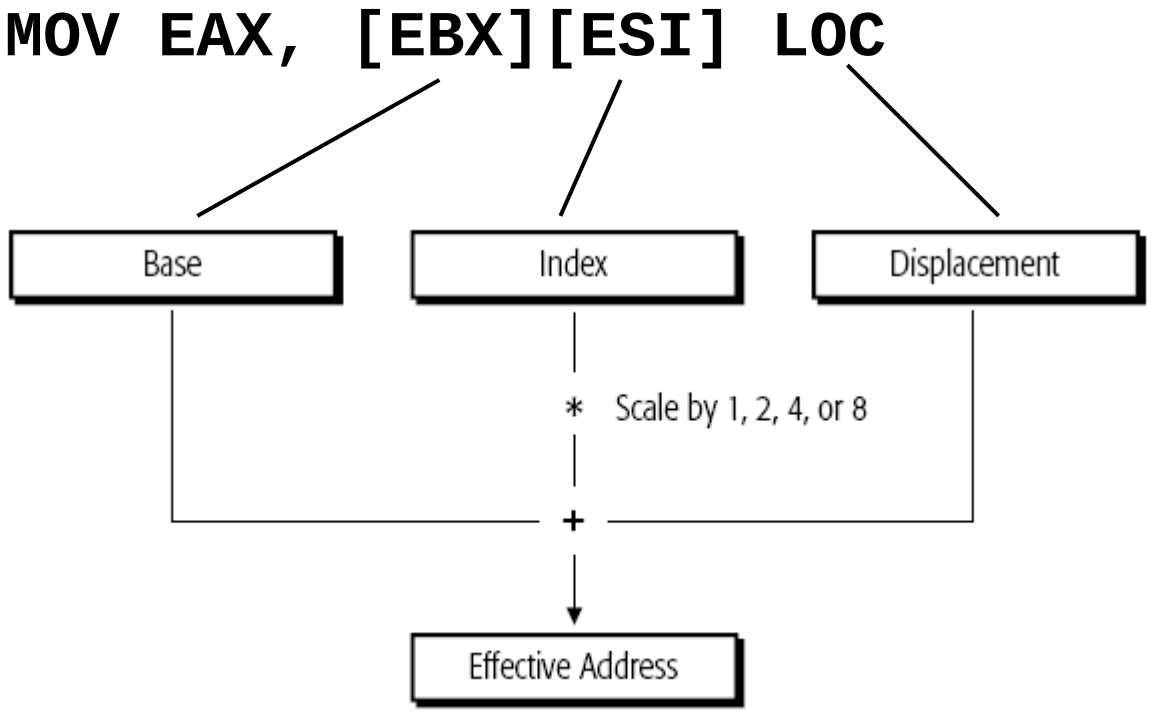
- Di solito si parla di **Effective Address** (EA)

Esempio

```
MOV EAX, [EBX][ESI] LOC
```

Effective address



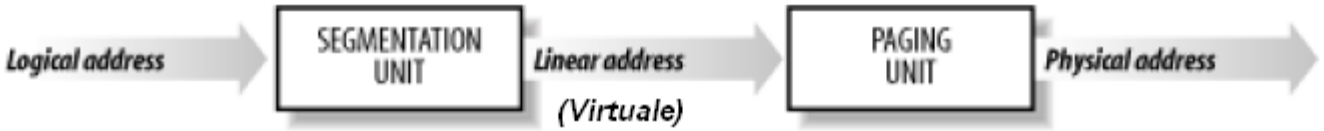


Memoria (definizioni)

- **Indirizzo lineare**: è l'indirizzo generato dall'unità di segmentazione, calcolato come

  `Indirizzo base del segmento + Effective Address`

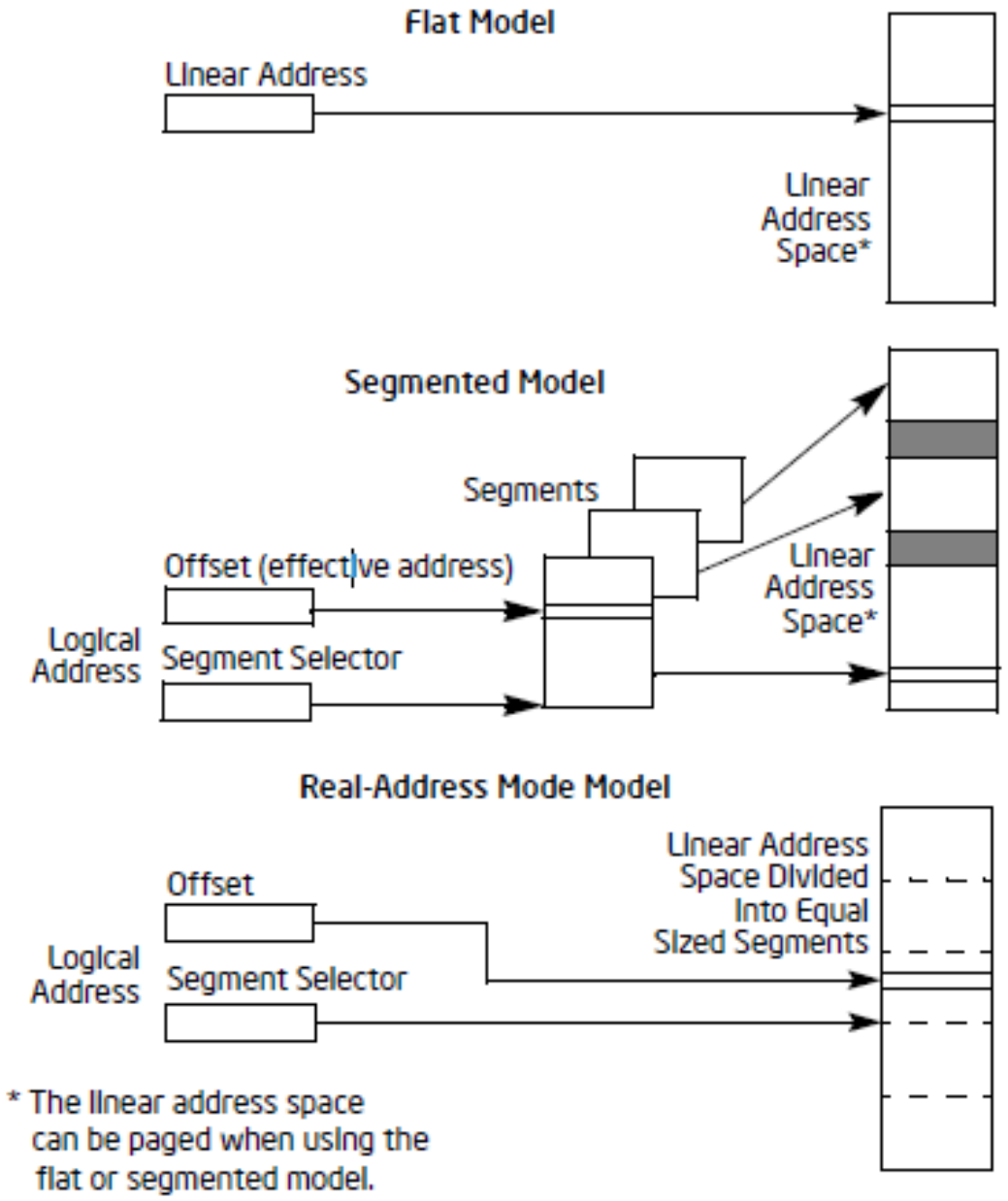
  - L'**`indirizzo base del segmento`** è ottenuto attraverso una tabella dei descrittori di segmento cui si accede tramite il selettore di segmento
- Nella terminologia x32/64 spesso i termini "lineare" e "virtuale" vengono usati in modo interscambiabile, in quanto a valle c'è la paginazione:
  - Se la paginazione è attiva, l'indirizzo lineare è l'indirizzo virtuale che viene mappato sulla memoria fisica dalla paginazione
  - Se la paginazione non è attiva l'indirizzo lineare (non è più virtuale e) si mappa direttamente sulla memoria fisica
  - Linux e Windows adottano questa terminologia



x32

TRE modi per la gestione della memoria

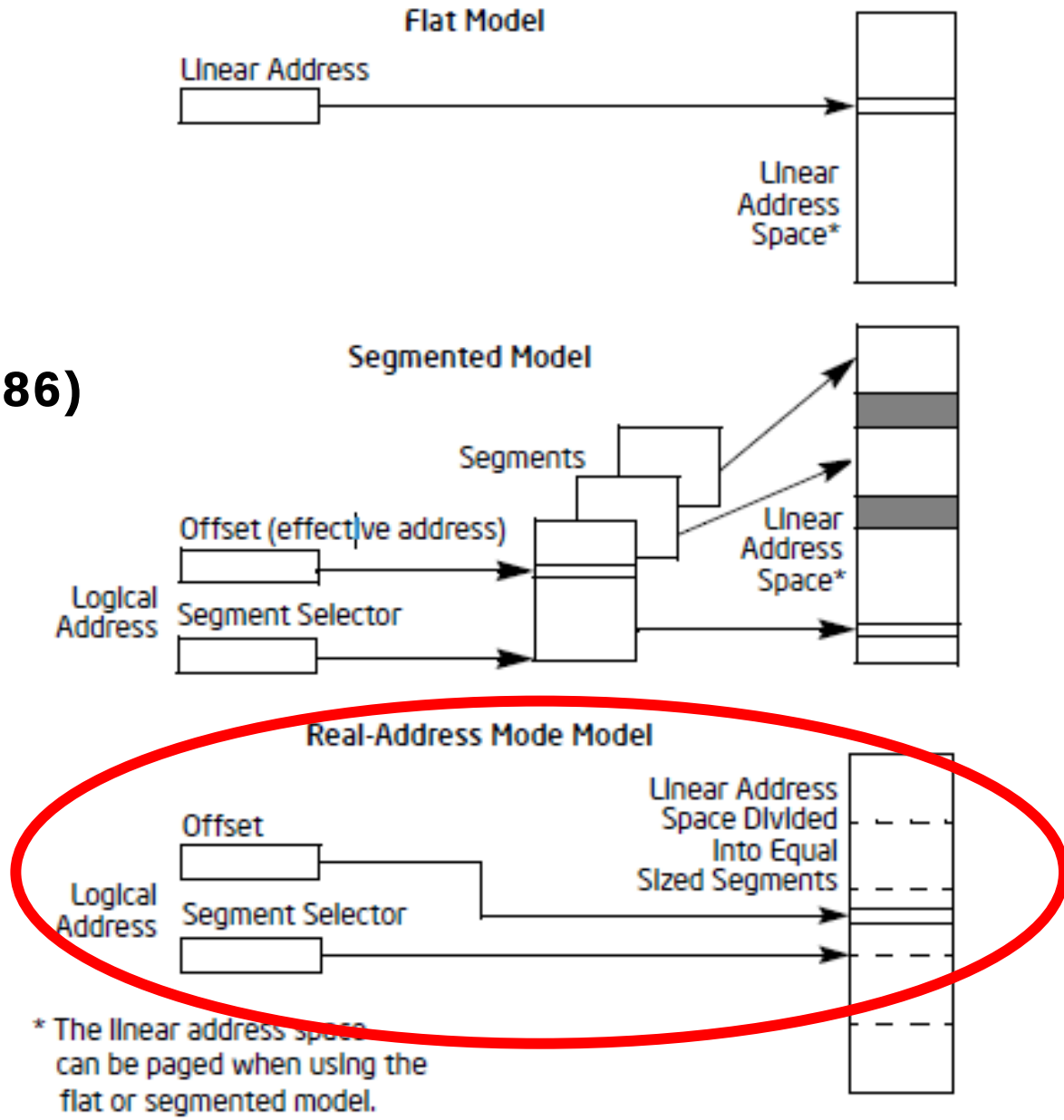
(Terminologia corrente)



x32

Modo reale

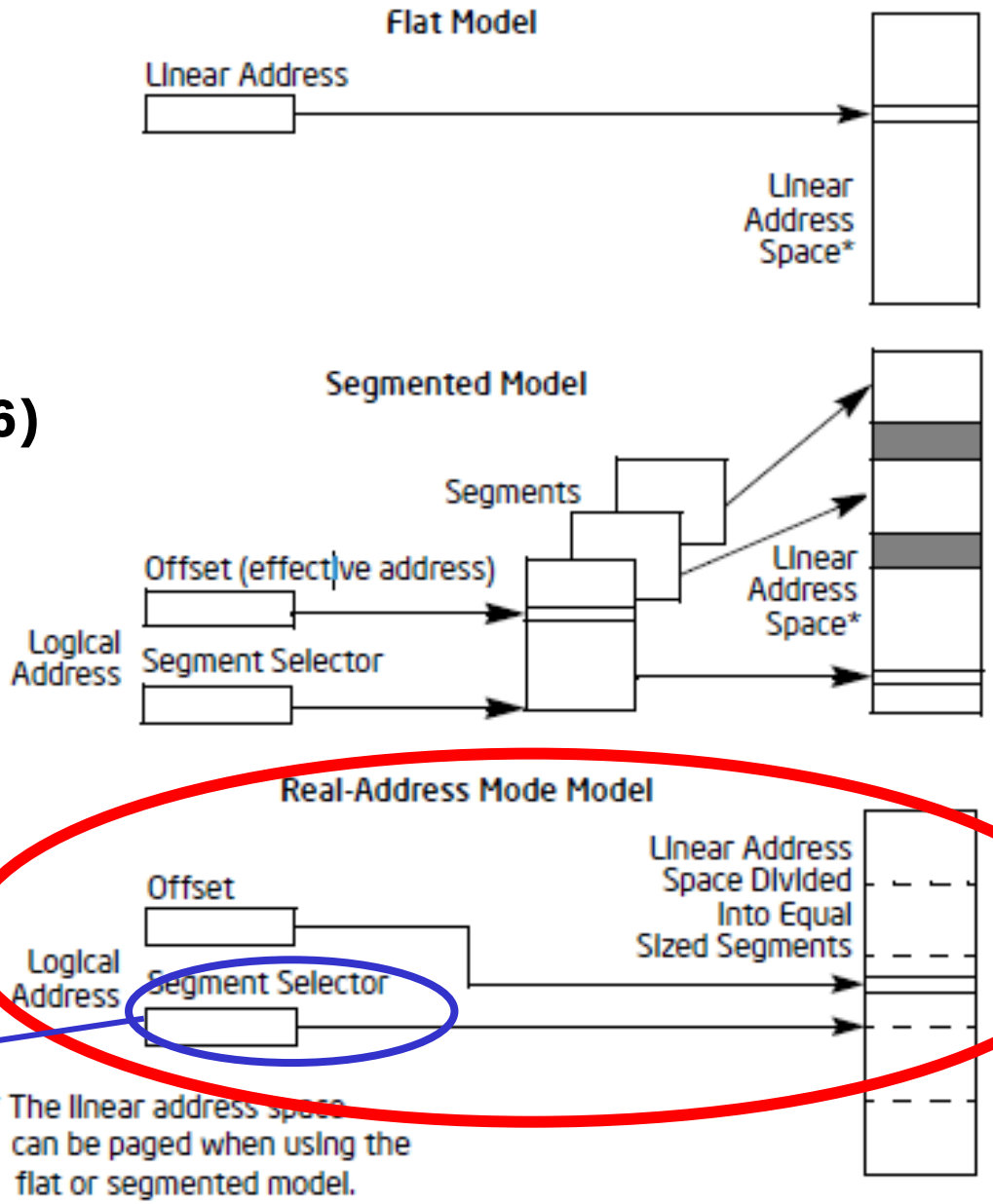
(per compatibilità con 8086)



x32

Modo reale

(per compatibilità con 8086)

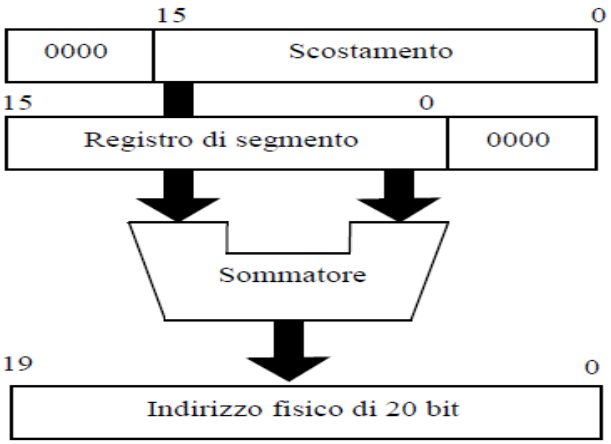


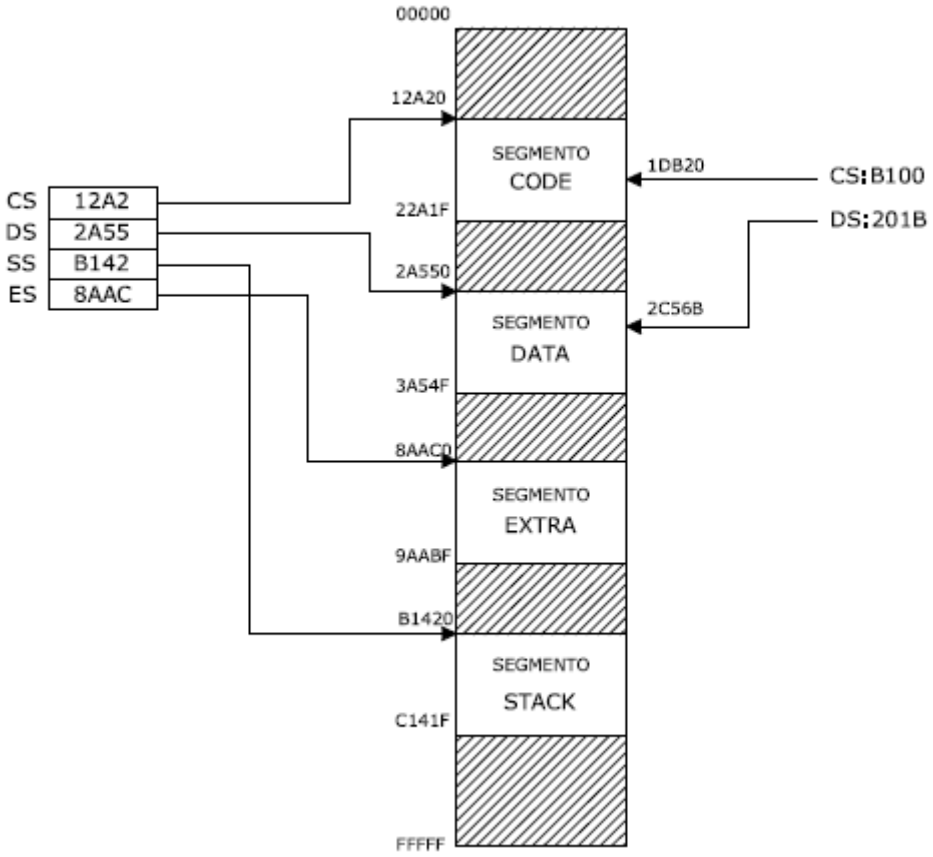
E questo che cos'è?

Un selettore o una base ?

Memoria 8086

- L'indirizzo lineare prodotto dall'unità di segmentazione viene preso come indirizzo fisico e passato direttamente alla memoria
- In "real address mode" x32 e A-64 si comportano come un 8086



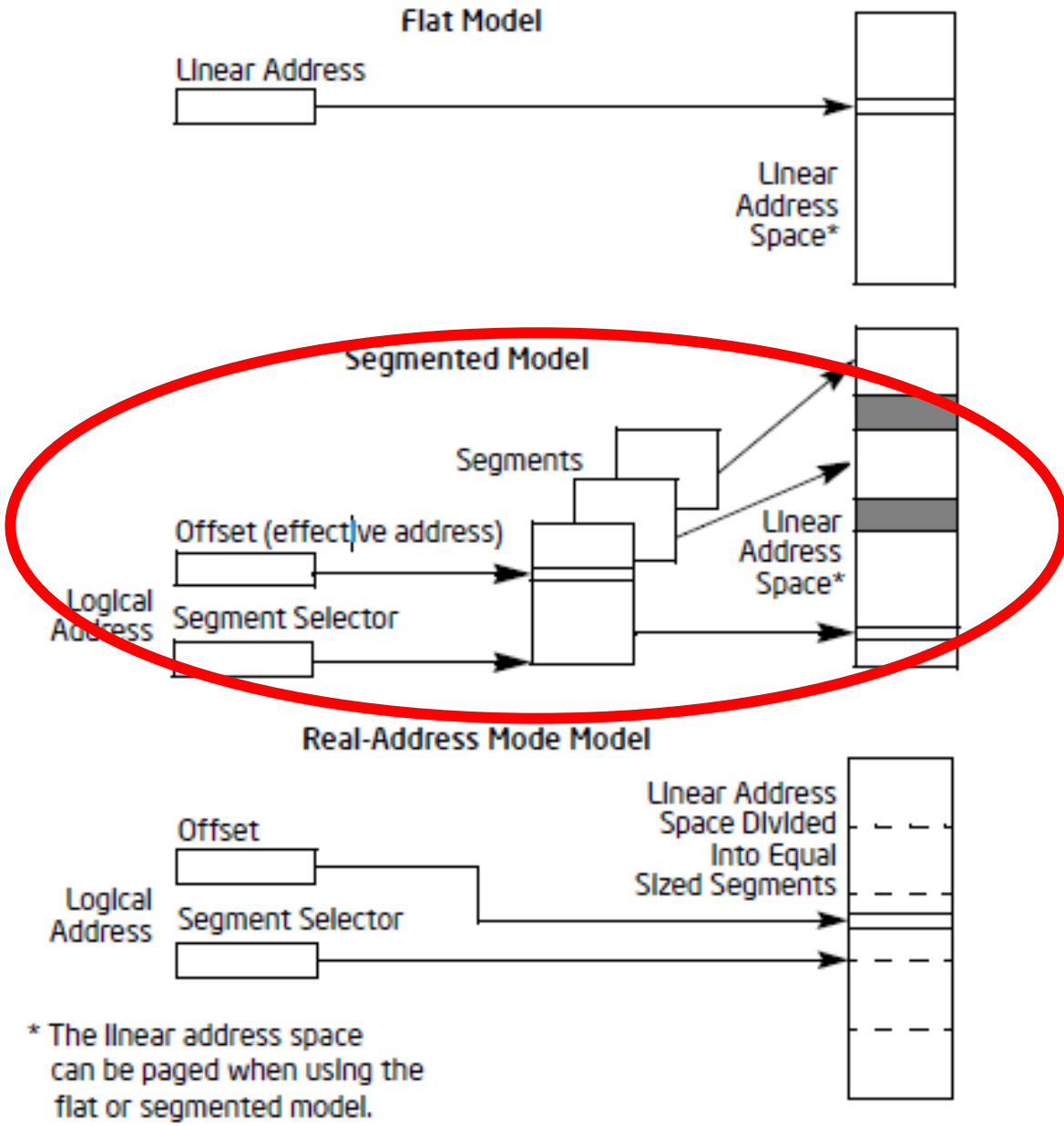


x32

Segmentata

(modo protetto)

Detto
“Modo Nativo”

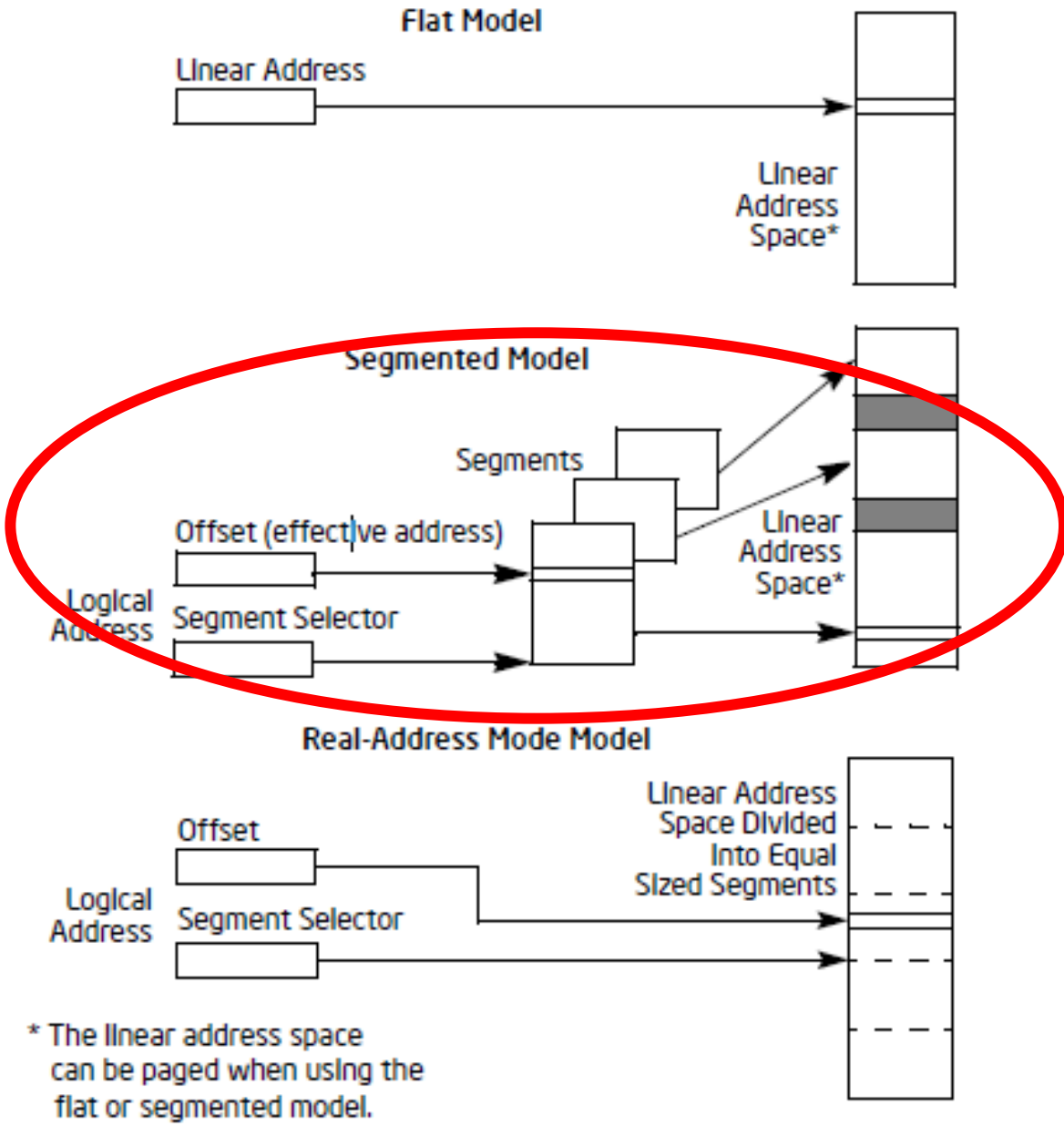


x32

Segmentata

(modo protetto)

Schema semplificato (manca roba)



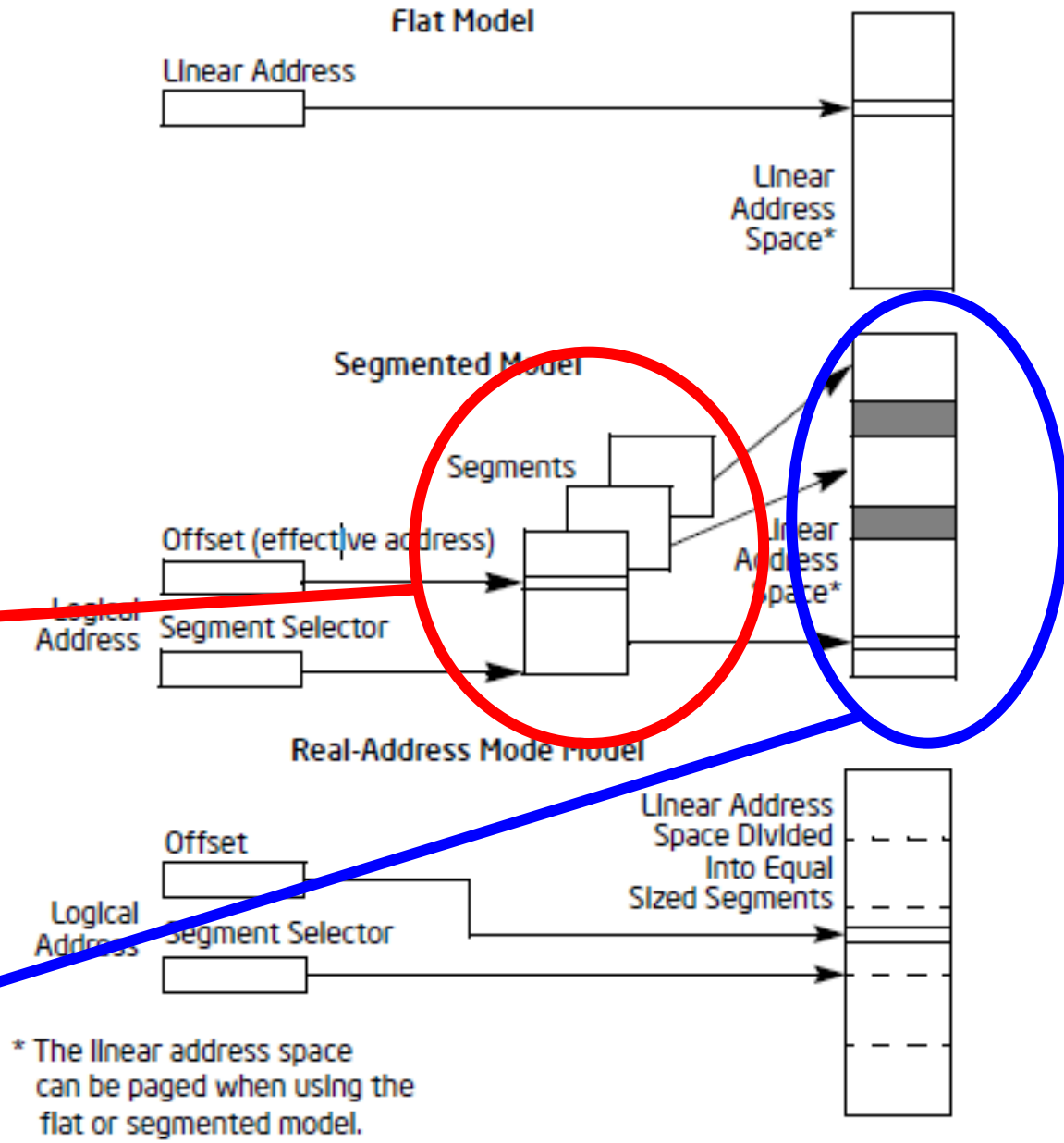
x32

Segmentata

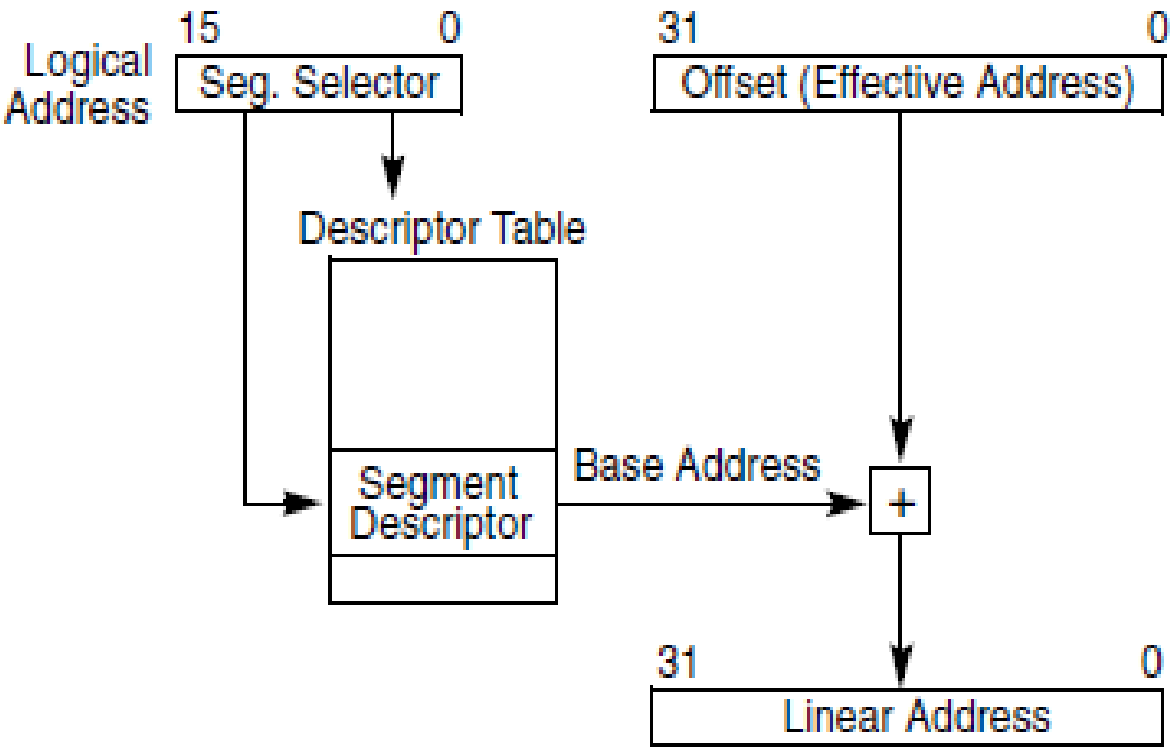
(modo protetto)

Spazio virtuale (64 TB)

Spazio fisico (4 GB)

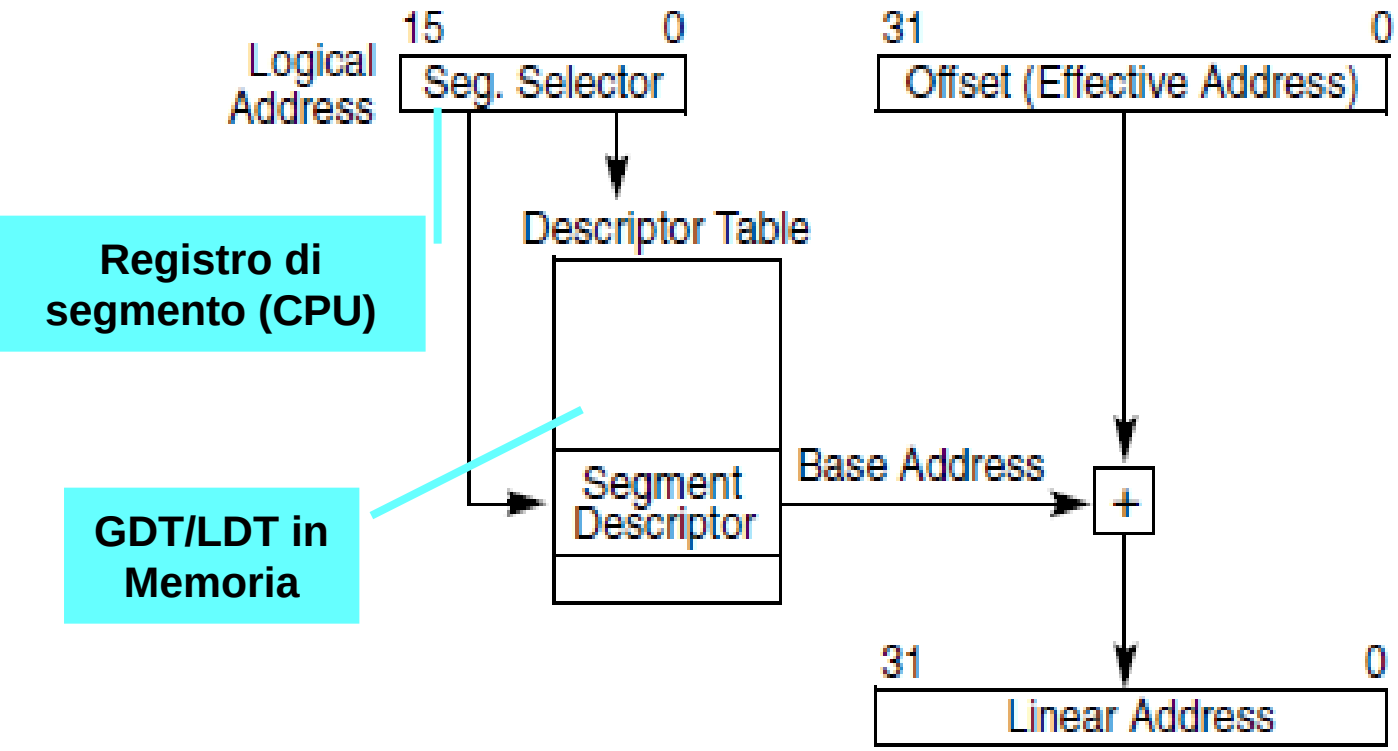


Memoria virtuale (solo) segmentata



La memoria virtuale segmentata è stata introdotta con il 286 (16 bit)
Spazio degli indirizzi virtuale: 1 GB
Spazio degli indirizzi lineare (fisico): 16 MB (24 linee di indirizzo)

… Memoria virtuale (solo) segmentata

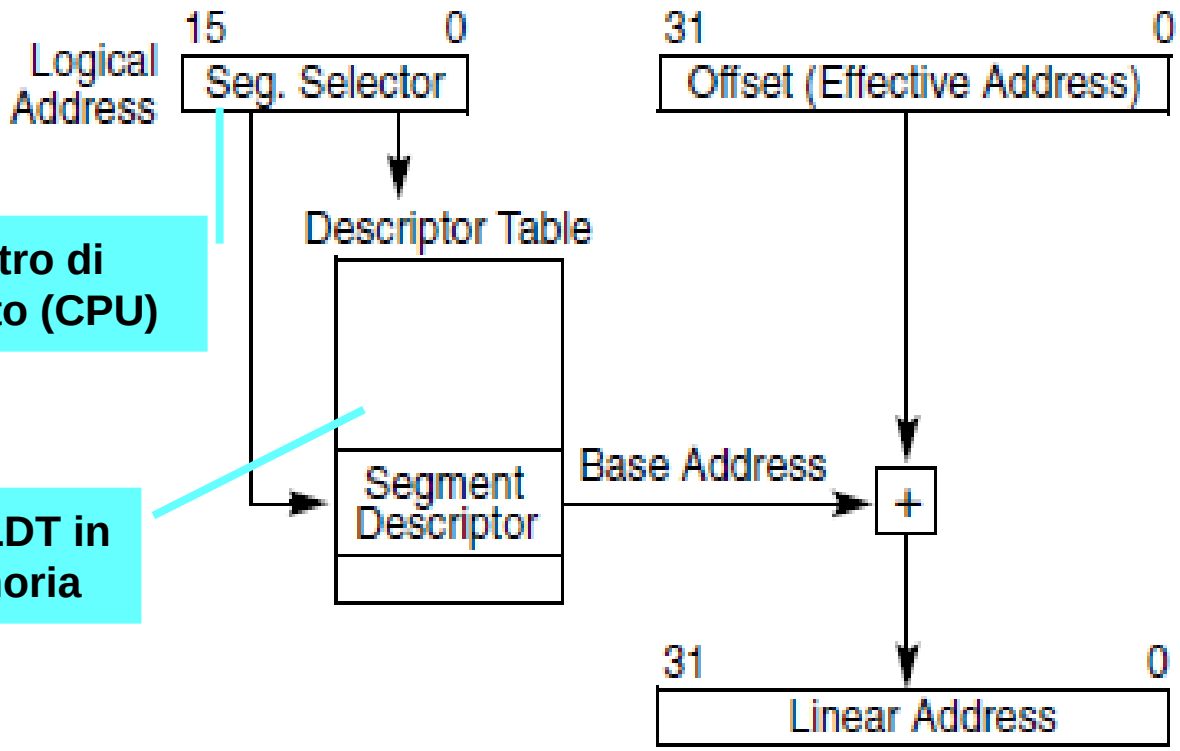


Le tabelle dei descrittori sono di 2 tipi: GDT e LDT

GDT: Globale, unica per tutto il sistema

LDT: Locale, associata al singolo task

…Memoria virtuale (solo) segmentata



Registro di segmento (CPU)

GDT/LDT in Memoria

x32

Spazio virtuale: 64 TB

Spazio lineare (fisico): 4 GB

Le tabelle dei descrittori sono di 2 tipi: GDT e LDT

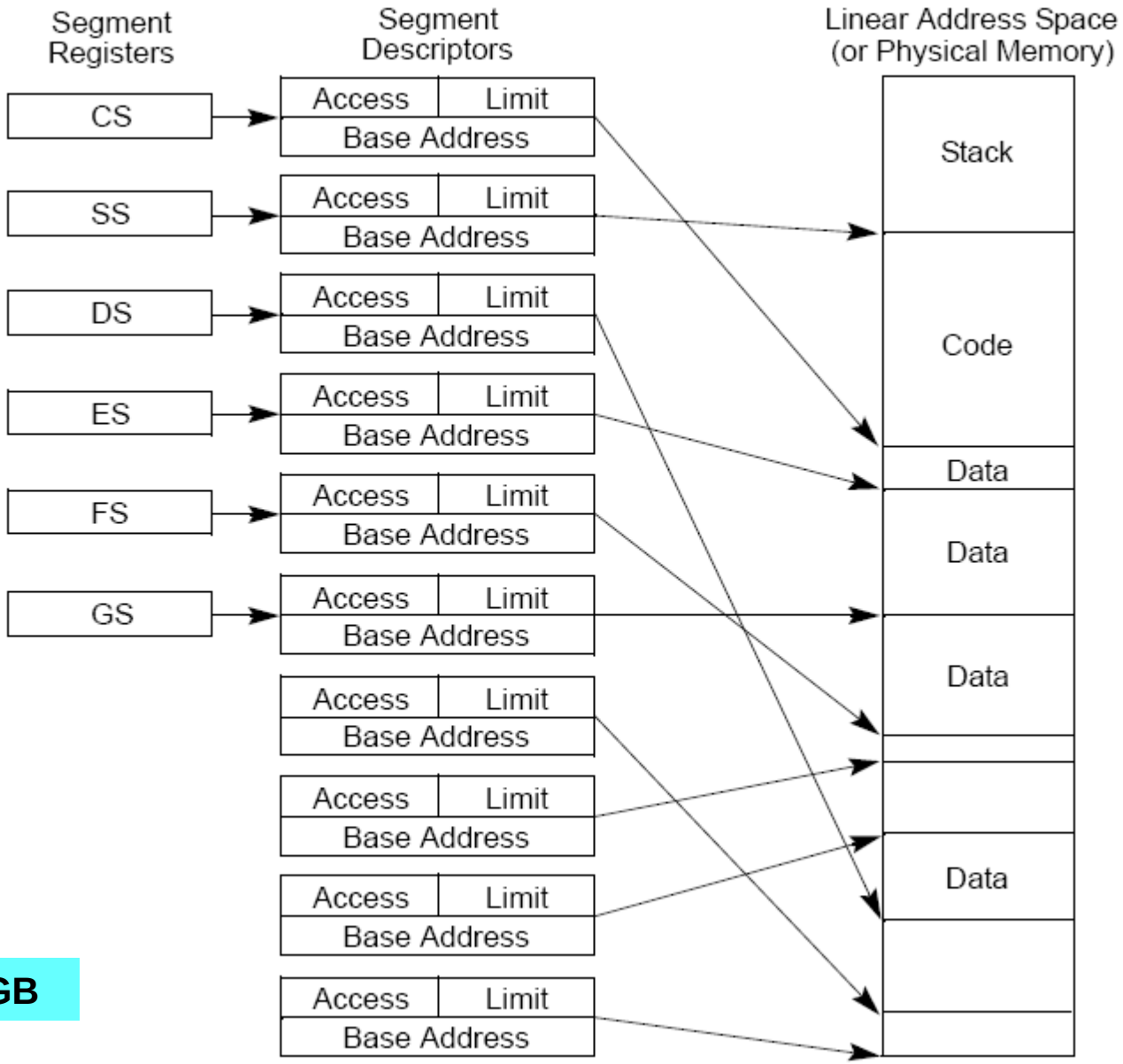
GDT: Globale, unica per tutto il sistema

LDT: Locale, associata al singolo task

x32

Dettaglio

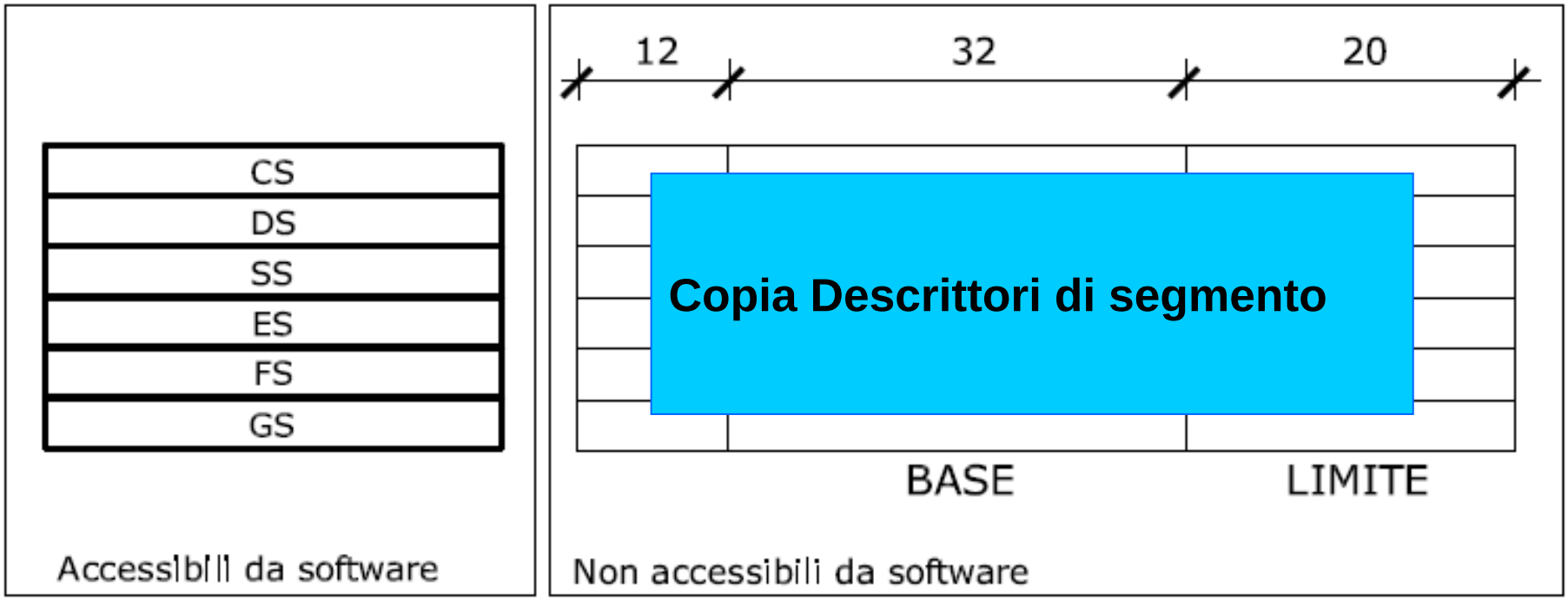
I registri di segmento contengono *selettori* che puntano a *descrittori* contenuti in apposite tabelle



Un segmento: fino a 4GB

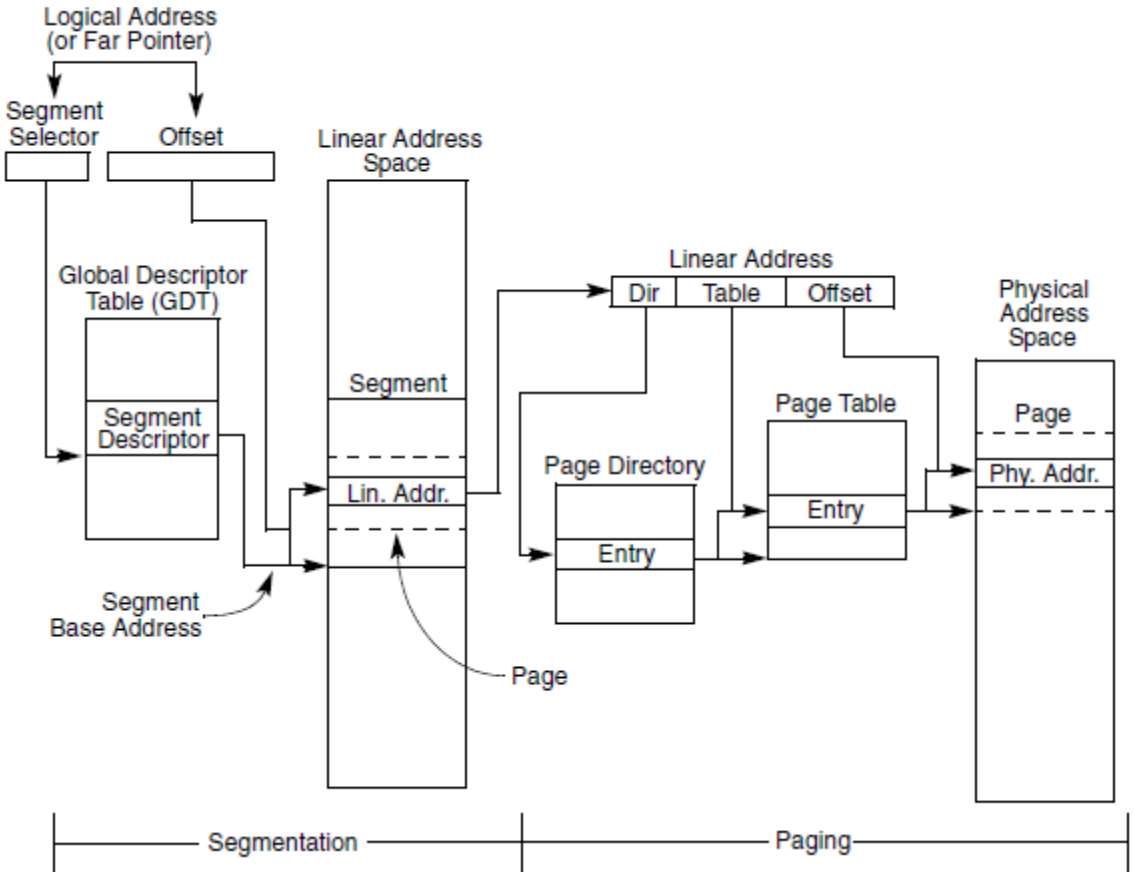
…c’è altro

- I registri di segmento hanno una parte nascosta (cache) nella quale vengono caricati i descrittori di segmento



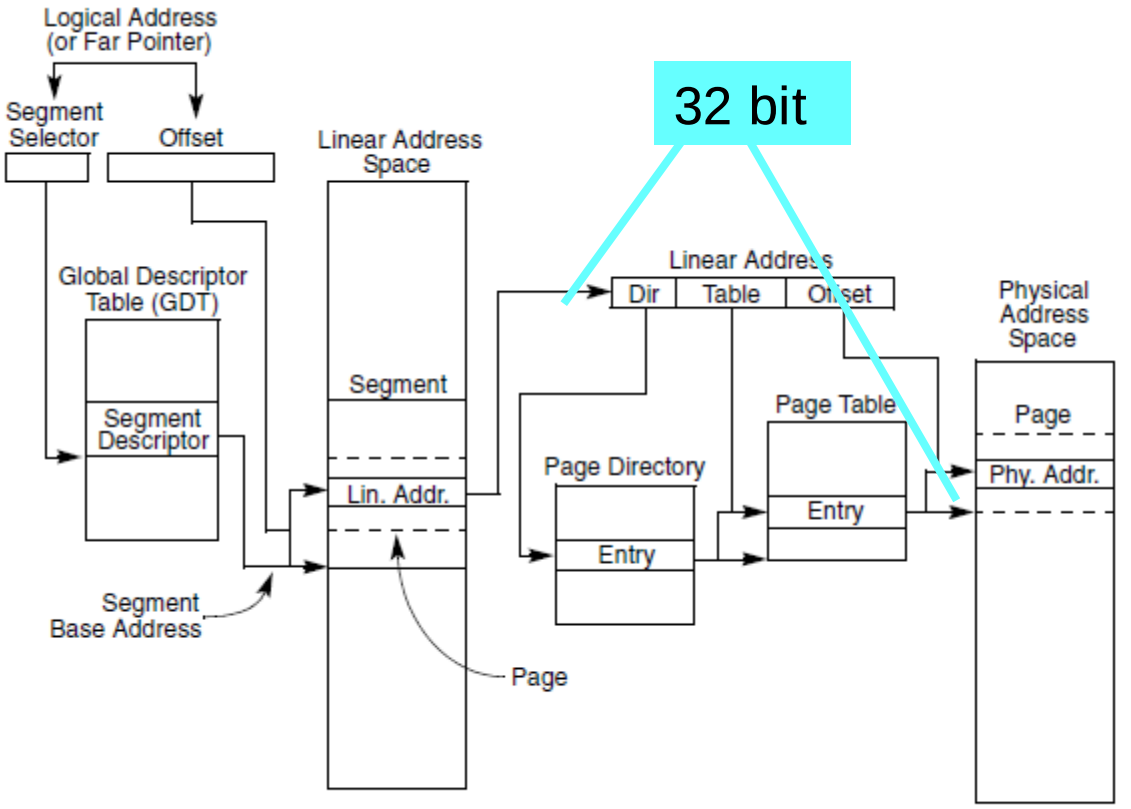
x32 paginazione (dal 386)

- Paginazione a valle della segmentazione
  - L'indirizzo (lineare) prodotto dall'unità di segmentazione viene assunto come indirizzo virtuale lineare, cioè pertinente ad una memoria virtuale lineare, a cui viene applicate la tecnica della paginazione.



x32 paginazione (dal 386)

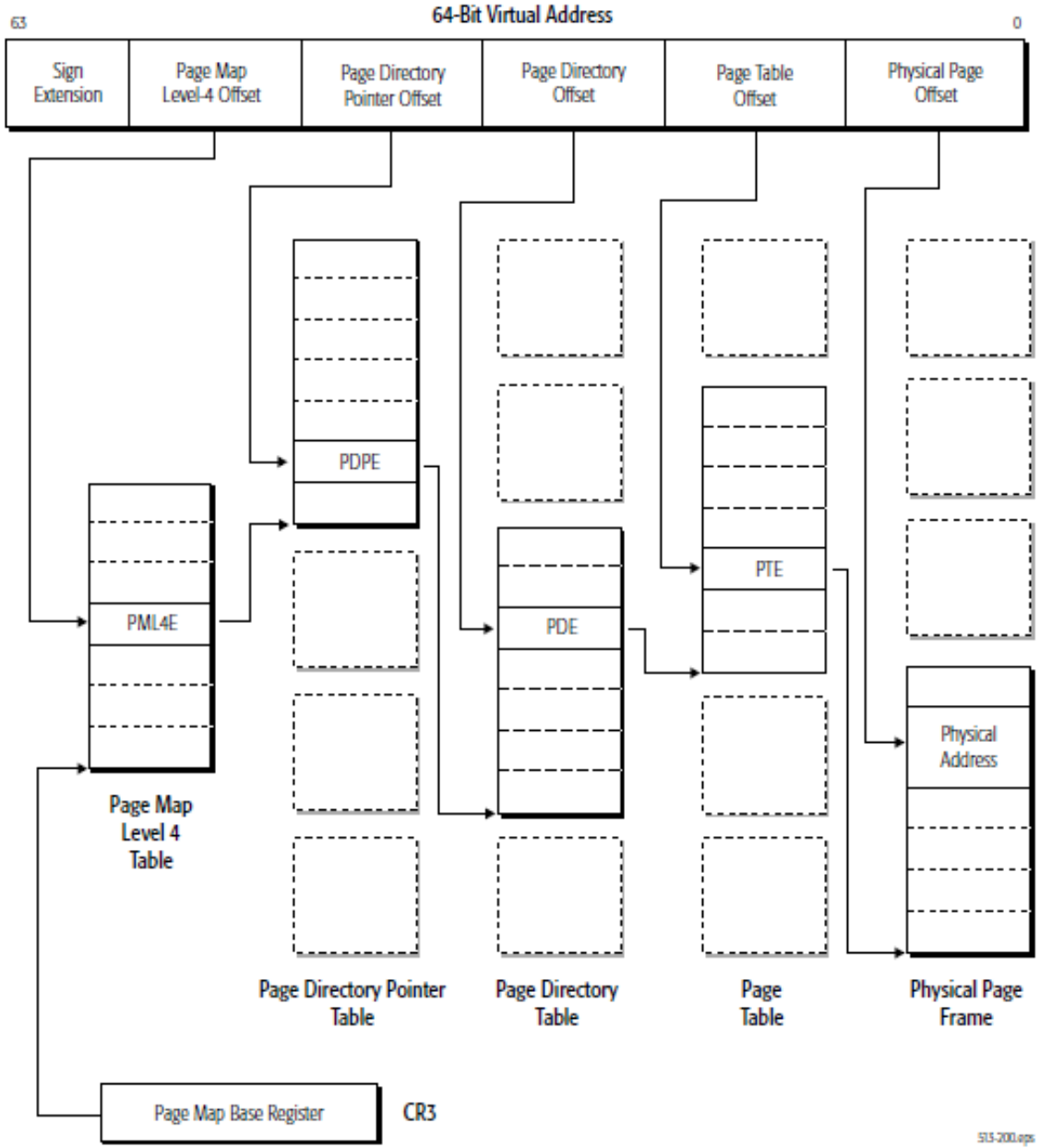
- Paginazione a valle della segmentazione
  - L'indirizzo (lineare) prodotto dall'unità di segmentazione viene assunto come indirizzo virtuale lineare, cioè pertinente ad una memoria virtuale lineare, a cui viene applicate la tecnica della paginazione.



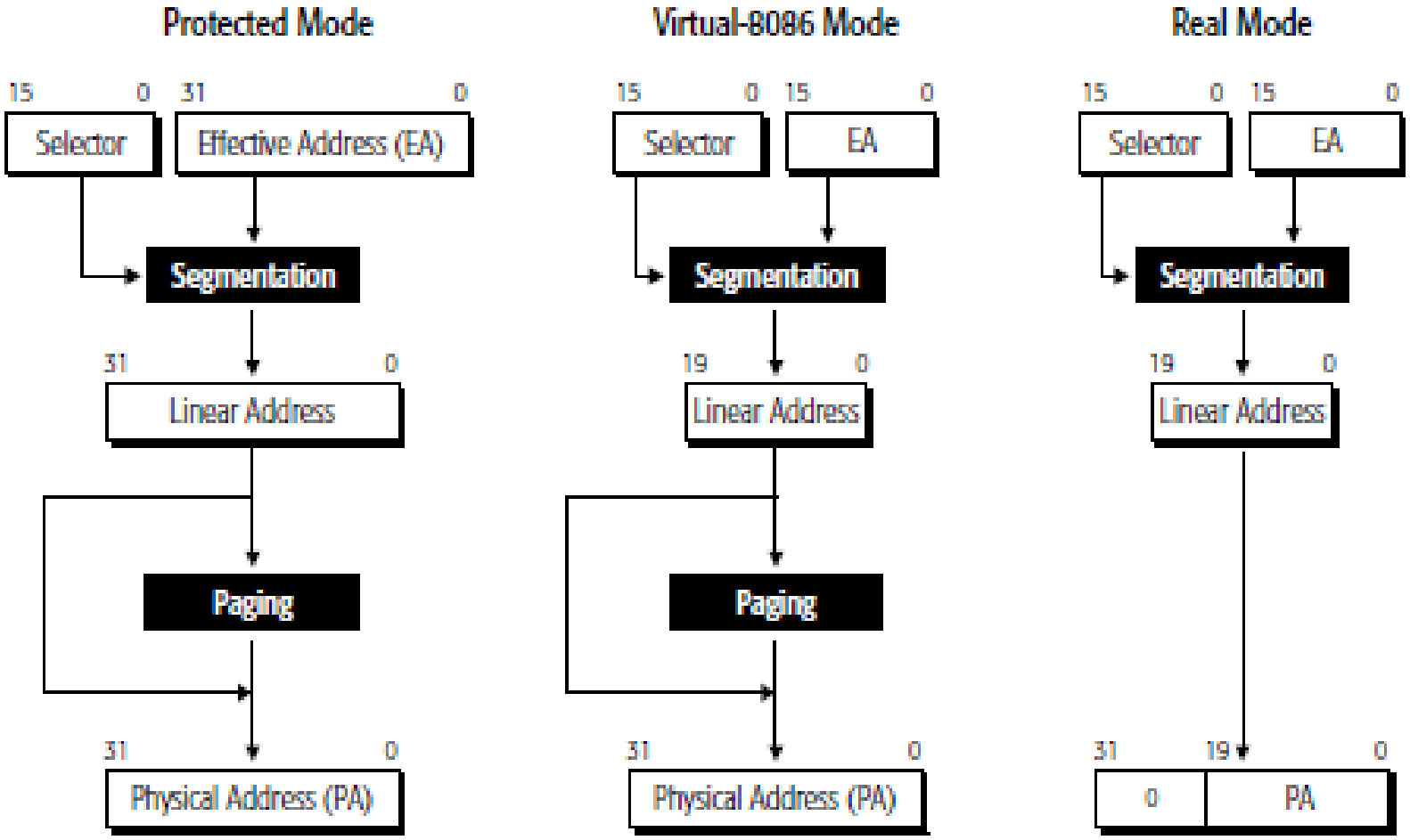
X32: Paginazione

- Nel 386 solo pagine di 4KB e traduzione a 2 livelli (come nella figura precedente)
- Correntemente le pagine possono essere di 4KB, 2MB o 4MB, i livelli di traduzione possono andare da 1 a 4 a seconda della dimensione della pagina
- Le tabelle di traduzione sono sempre di 4KB e devono essere allineate a confini di 4KB
- Le pagine fisiche devono essere allineate a confini di 4KB, 2MB o 4MB a seconda della loro dimensione.

x64: possibile paginazione



Traduzione x32

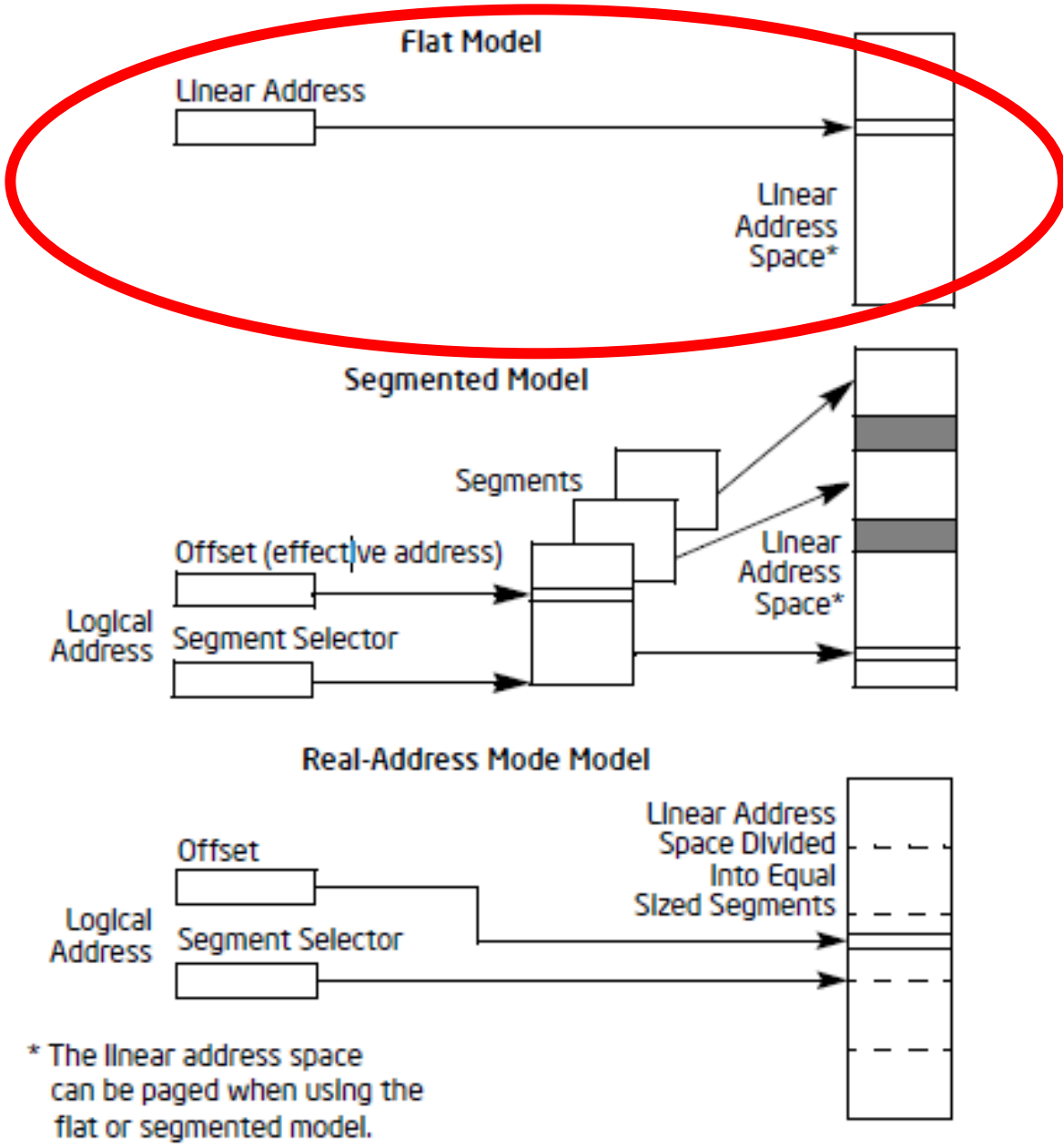


x32

Flat

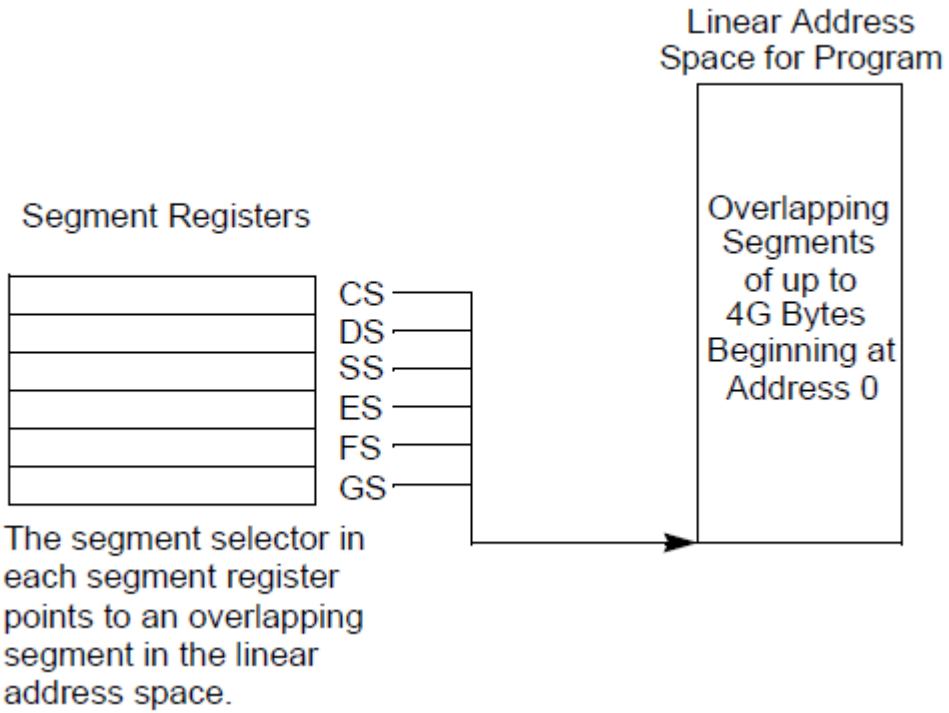
(modo protetto)

E’ un trucco

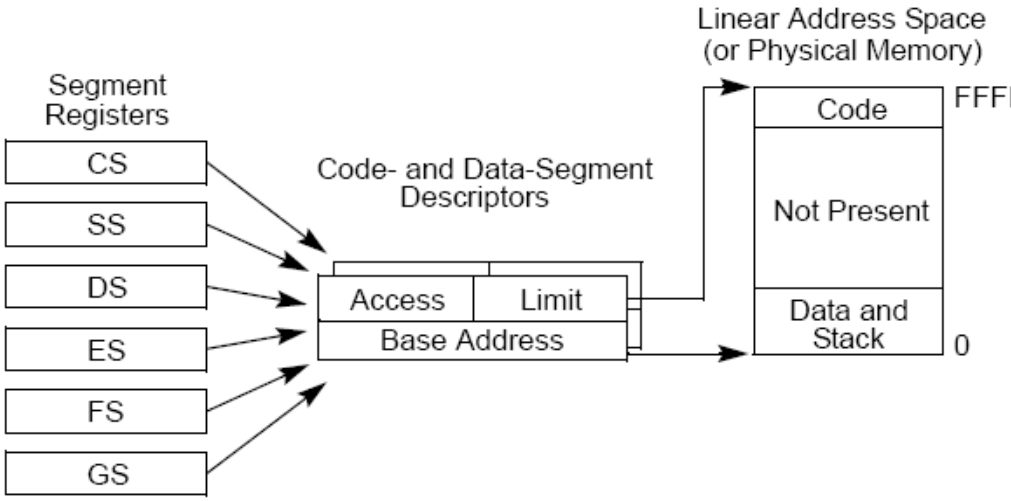


X32 Modello Piatto

- Come si ottiene il modello piatto



Il trucco per dare a x32 il modello Flat



Flat (puro)

Almeno due segmenti (uno per il codice e uno per dati e stack)

Ampi 4GB

Completamente **sovrapposti** (è quasi rinunciare alla protezione)

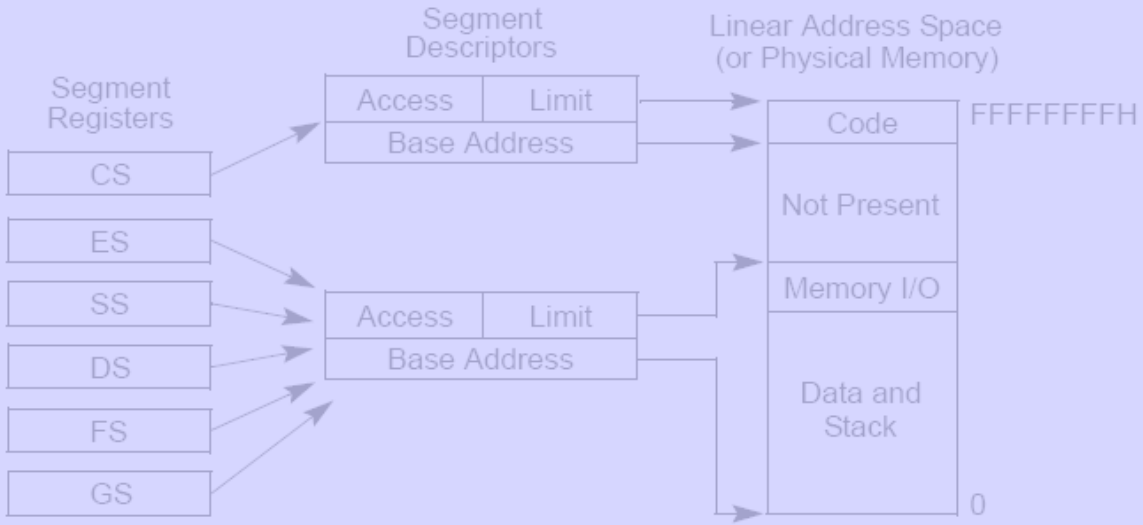
L'indirizzo lineare è l'effective address

Flat protetto

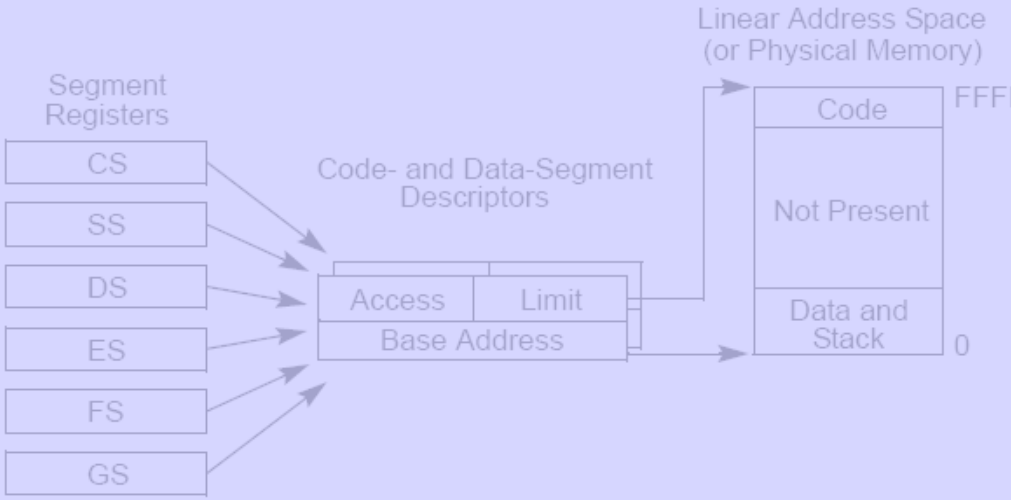
Almeno due segmenti (uno per il codice e uno per dati e stack)

Ampi quanto basta

Separati (si mantiene la protezione; è una memoria virtuale "mutilata")



Il trucco per dare a x32 il modello Flat



Flat (puro)

Almeno due segmenti (uno per il codice e uno per dati e stack)

Ampi 4GB

Completamente **sovrapposti** (è quasi rinunciare alla protezione)

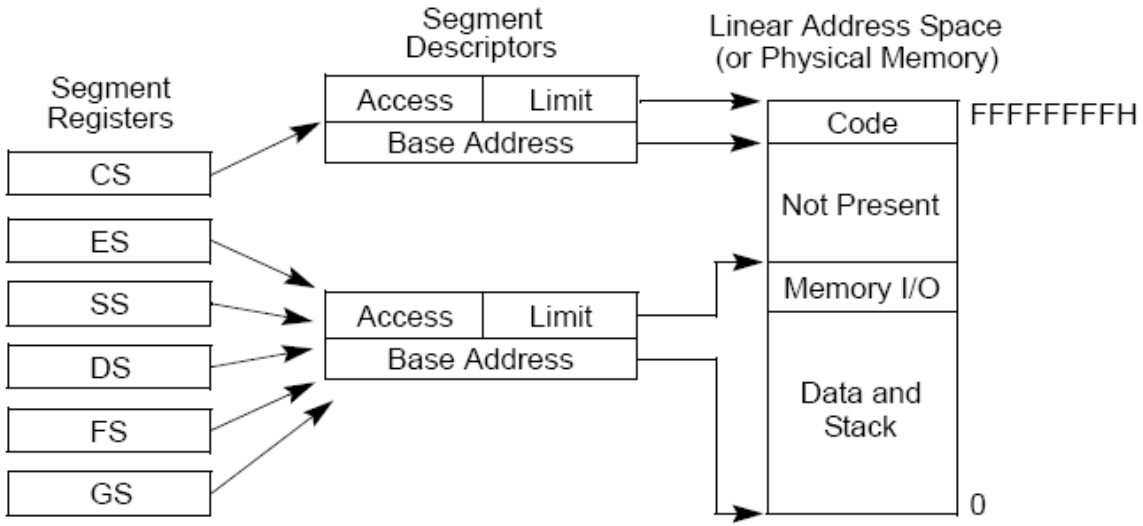
L'indirizzo lineare è l'effective address

Flat protetto

Almeno due segmenti (uno per il codice e uno per dati e stack)

Ampi quanto basta

Separati (si mantiene la protezione; è una memoria virtuale "mutilata")



Notare che

- Per realizzare il modello flat (puro) su x32
  - Tutti i segmenti devono avere una base pari a 0
  - I segmenti possono avere la dimensione limite di 4 GB
  - La segmentazione NON è (non può essere) disabilitata. L'indirizzo lineare generato coincide con l'effective address

- **Praticamente tutti i sistemi operativi usano il modello flat**

- x64 implementa direttamente un modello di memoria flat
  - In modo 64-bit la base dei segmenti è trattata automaticamente come fosse 0 e il limite del segmento viene ignorato.
  - Con l'effective address si può accedere a tutto lo spazio di indirizzi (lineare)

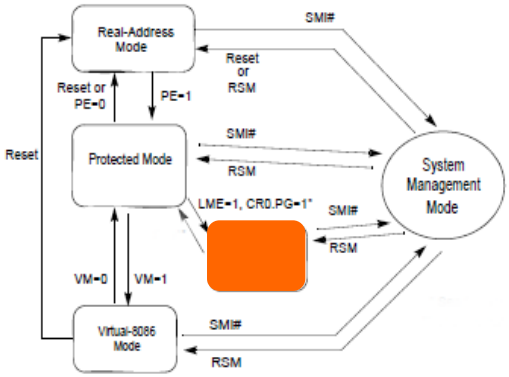
Notare che

- Per realizzare il modello flat (puro)in x32
  - Tutti i segmenti devono avere una base pari a 0
  - I segmenti possono avere la dimensione limite di 4 GB
  - La segmentazione NON è (non può essere) disabilitata. L'indirizzo lineare generato coincide con l'effective address

- **Praticamente tutti i sistemi operativi usano il modello flat**

- x64 implementa direttamente un modello di memoria flat
  - In modo 64-bit la base dei segmenti è trattata automaticamente come fosse 0 e il limite del segmento viene ignorato.
  - Con l'effective address si può accedere a tutto lo spazio di indirizzi (lineare)

Segmentazione in Modo x32e



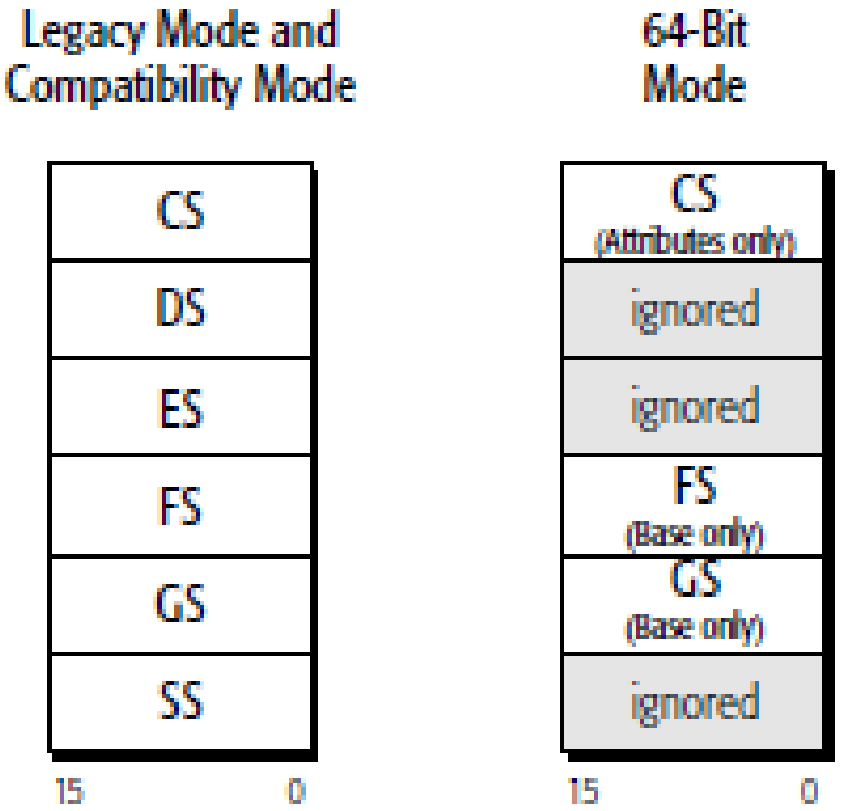
- **Compatibility Mode**
  - Esattamente come per x32

- **64-Bit Mode**
  - Segmentazione (quasi) disabilitata: la base dei segmenti associati a CS, DS, SS, ES è presa come se fosse sempre ZERO
  - Non vengono fatti i check di superamento dei limiti del segmento (*)
  - Ne deriva un indirizzamento lineare (da 0 fino a 2^{64})
  - L'effettiva dimensione della memoria fisica può essere diversa da modello a modello (istruzione CPUID)

---

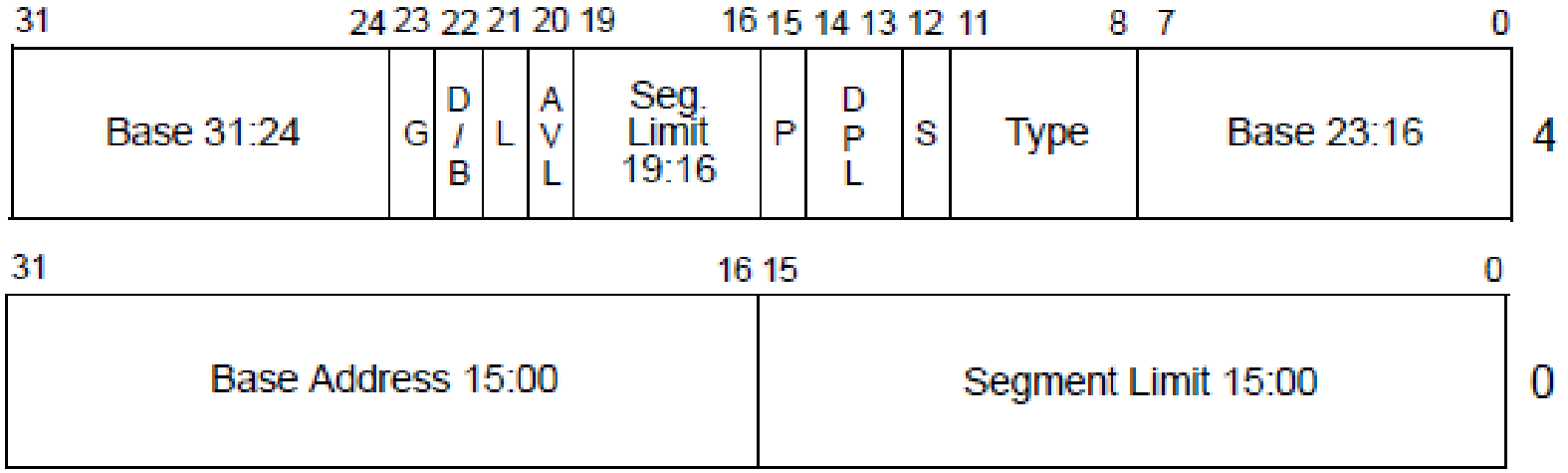
* Non è del tutto vero: nei processori AMD il controllo del limite è stato reintrodotto nelle versioni più recenti a 64 bit.

I registri di segmento



- DS, ES e SS vengono ignorati nel modo 64-bit
- FS e GS usati a volte nel calcolo dell'EA (aggiungendosi ai termini visti in precedenza) [wrap around]

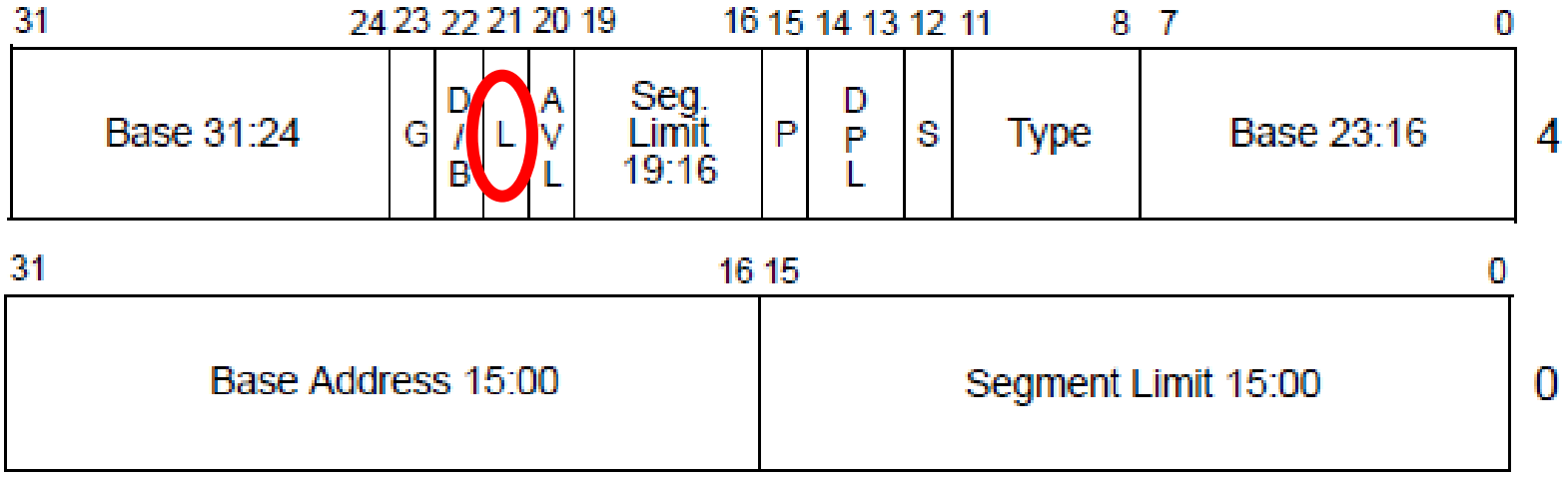
Dettaglio un descrittore di seg



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

Prima era sempre 0

32/64 bit

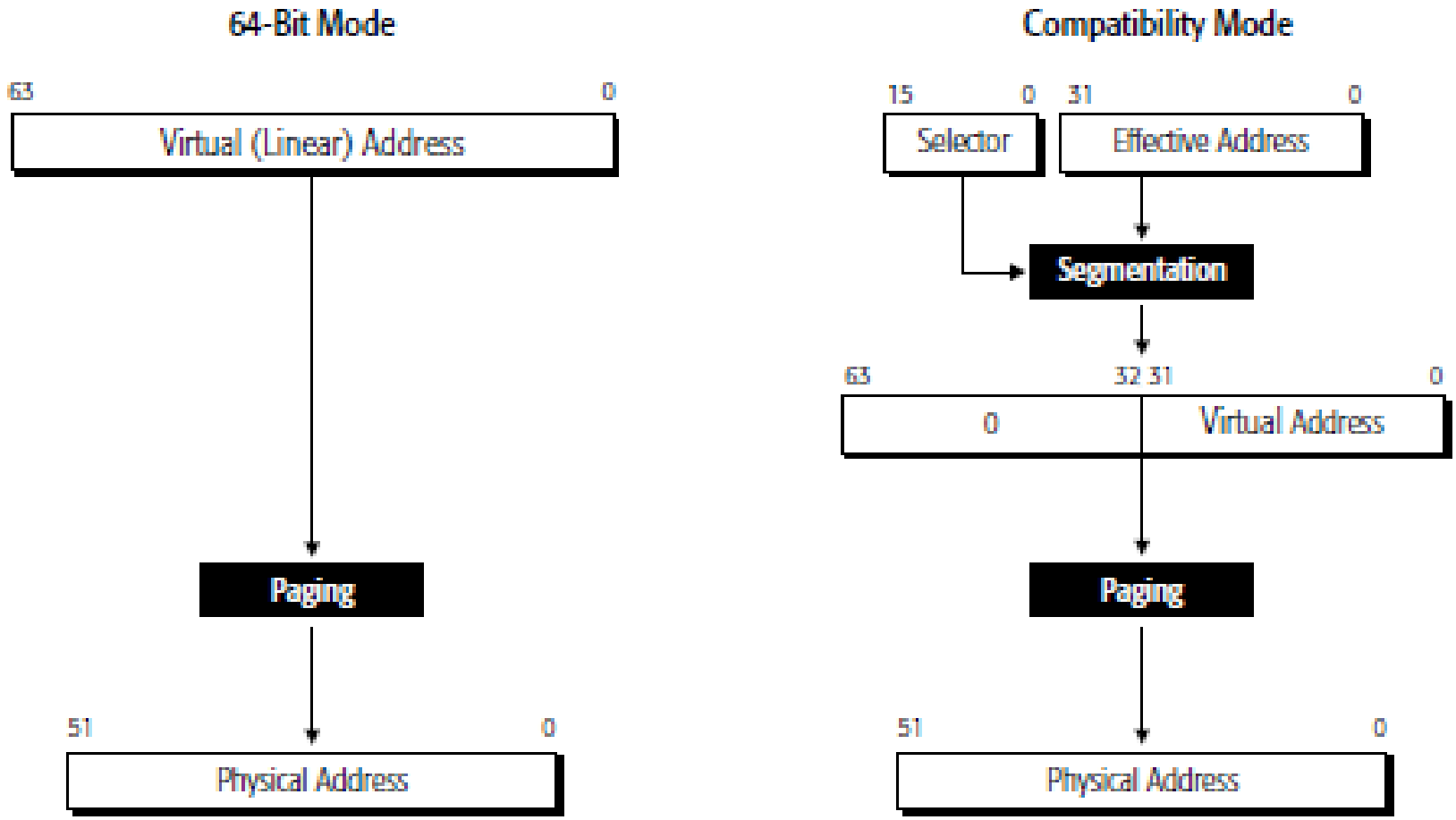


In modo x32e, (per i segmenti di codice):

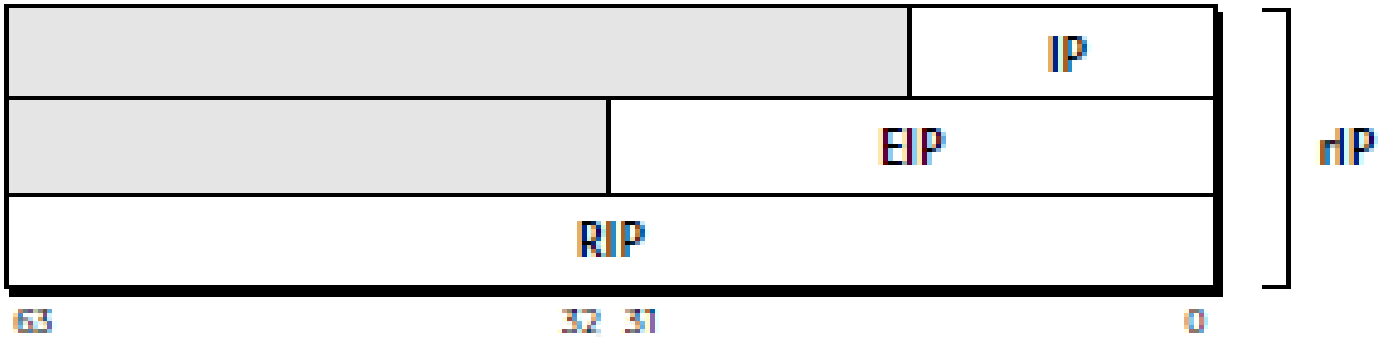
L=1 indica che il segmento è da eseguire in modo 64 bit

L=0 indica che deve essere eseguito in modo compatibile

Traduzione in modo x32 esteso



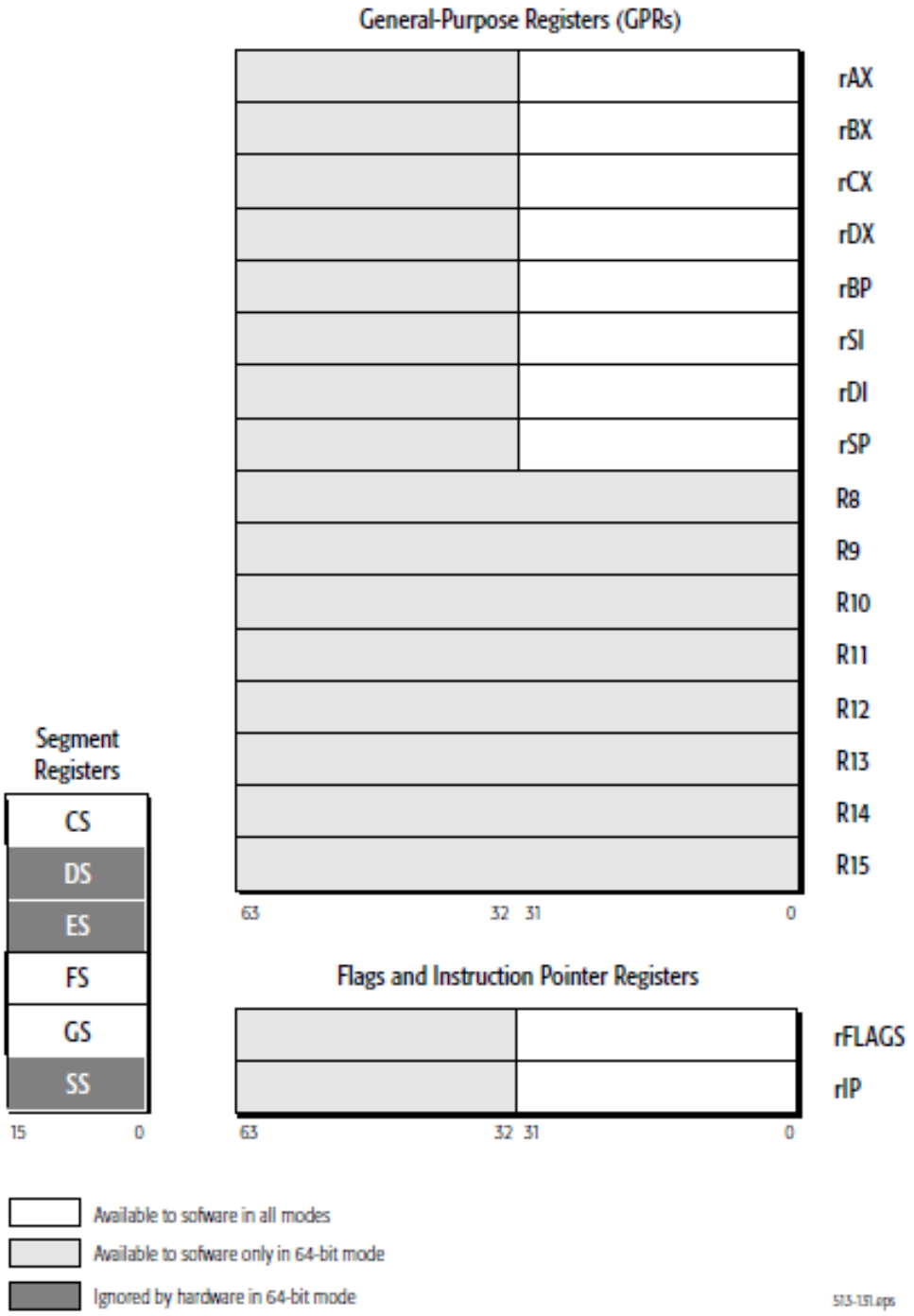
Instruction Pointer



Program Counter (logico) = CS : rIP

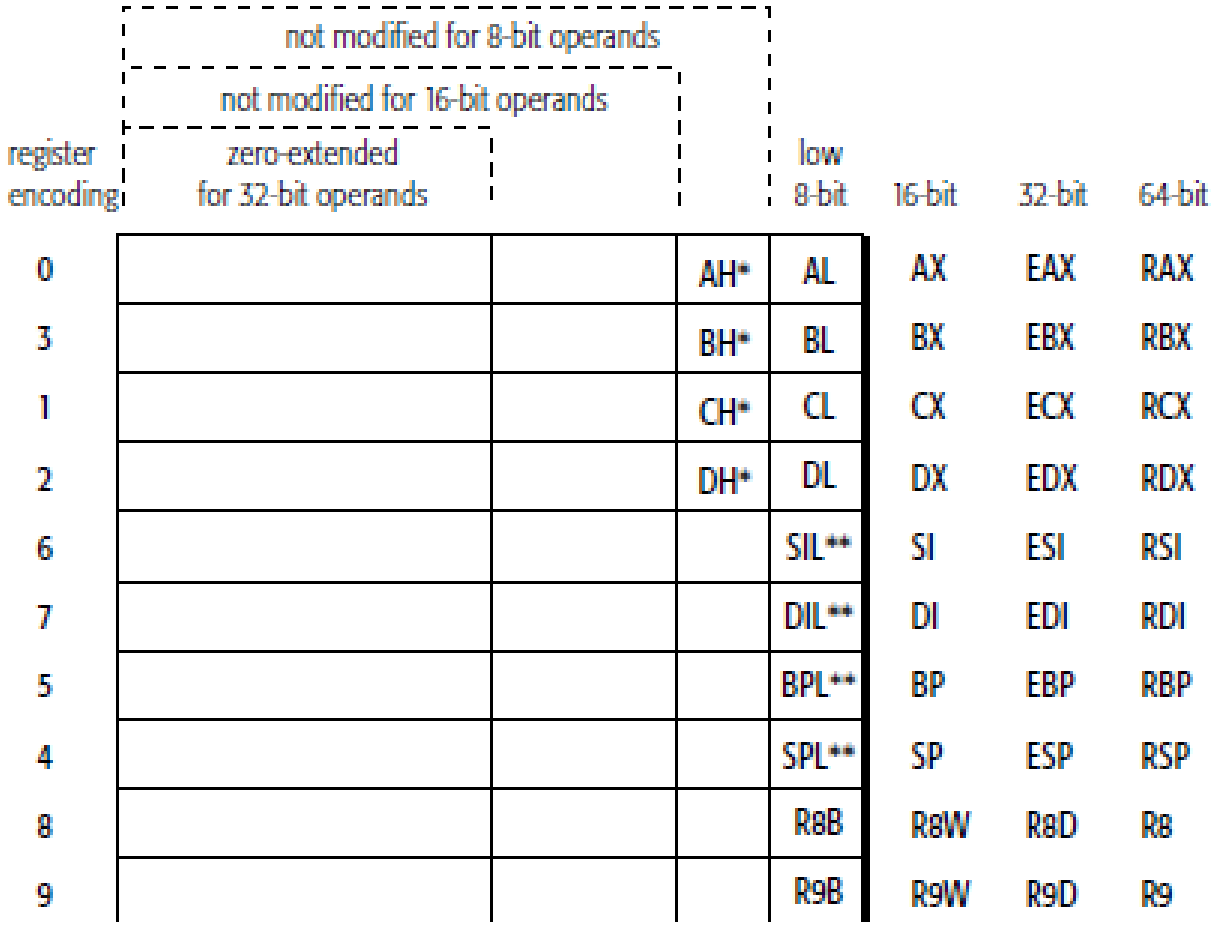
- **16/32 bit: CS : IP/EIP**
- **64 bit: RIP**
  - **In modo 64 bit CS dà sempre 0 come base del segmento**

Registri


General-Purpose Registers (GPRs)
rAX
rBX
rCX
rDX
rBP
rSI
rDI
rSP
R8
R9
R10
R11
R12
R13
R14
R15
63
32 31
0
Segment Registers
CS
DS
ES
FS
GS
SS
15
0
Flags and Instruction Pointer Registers
rFLAGS
rIP
63
32 31
0
Available to software in all modes
Available to software only in 64-bit mode
Ignored by hardware in 64-bit mode

Prefisso REX

Per trattare registri a 64 bit



register encoding	not modified for 8-bit operands	not modified for 16-bit operands	zero-extended for 32-bit operands		low 8-bit	16-bit	32-bit	64-bit
0				AH*	AL	AX	EAX	RAX
3				BH*	BL	BX	EBX	RBX
1				CH*	CL	CX	ECX	RCX
2				DH*	DL	DX	EDX	RDX
6					SIL**	SI	ESI	RSI
7					DIL**	DI	EDI	RDI
5					BPL**	BP	EBP	RBP
4					SPL**	SP	ESP	RSP
8					R8B	R8W	R8D	R8
9					R9B	R9W	R9D	R9

* Not addressable when a REX prefix is used.

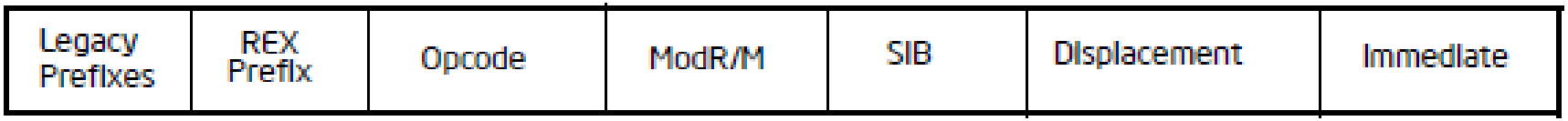
** Only addressable when a REX prefix is used.

Esempio (effetto REX)

- Mostriamo la somma di 2 registri, nella versione a 64, 32, 16, 8 bit
  - 64: ADD RBX,RAX
  - 32: ADD EBX,EAX
  - 16: ADD BX,AX
  - 8: ADD BL,AL

Esempio (effetto REX)

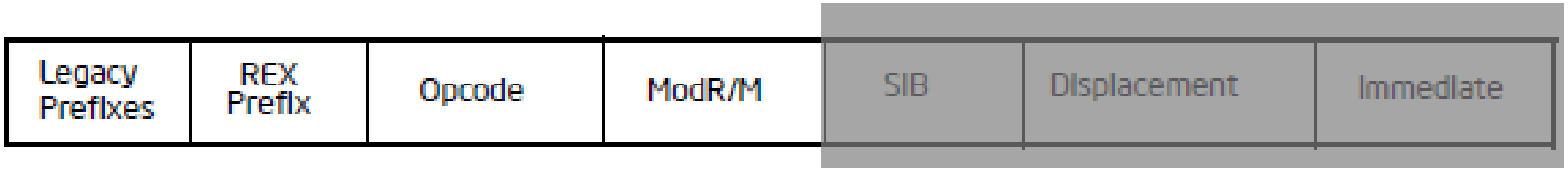
- Mostriamo la somma di 2 registri, nella versione a 64, 32, 16, 8 bit.
  - 64: ADD RBX,RAX
  - 32: ADD EBX,EA
  - 16: ADD BX,AX
  - 8: ADD BL,AL



Formato generico x32e

Esempio (effetto REX)

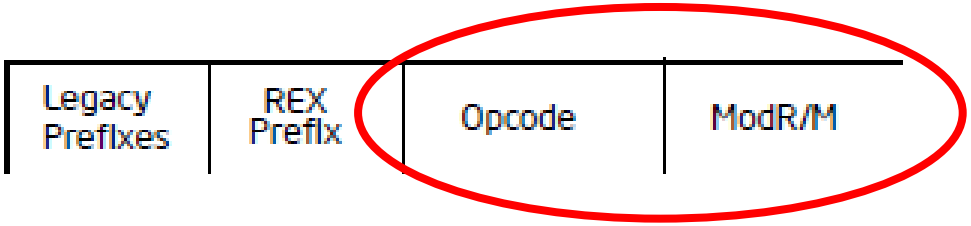
- Mostriamo la somma di 2 registri, nella versione a 64, 32, 16, 8 bit
  - 64: ADD RBX,RAX
  - 32: ADD EBX,EAX
  - 16: ADD BX,AX
  - 8: ADD BL,AL



I campi in grigio non ci sono (operazione tra registri)

Esempio (effetto REX)

- Mostriamo la somma di 2 registri, nella versione a 64, 32, 16, 8 bit
  - 64: ADD RBX,RAX
  - 32: ADD EBX,EAX
  - 16: ADD BX,AX
  - 8: ADD BL,AL



Codifica ADD (repertorio x32), pari a:

- 00C3 per 8 bit
- 01C3 per 16, 32 e 64 bit

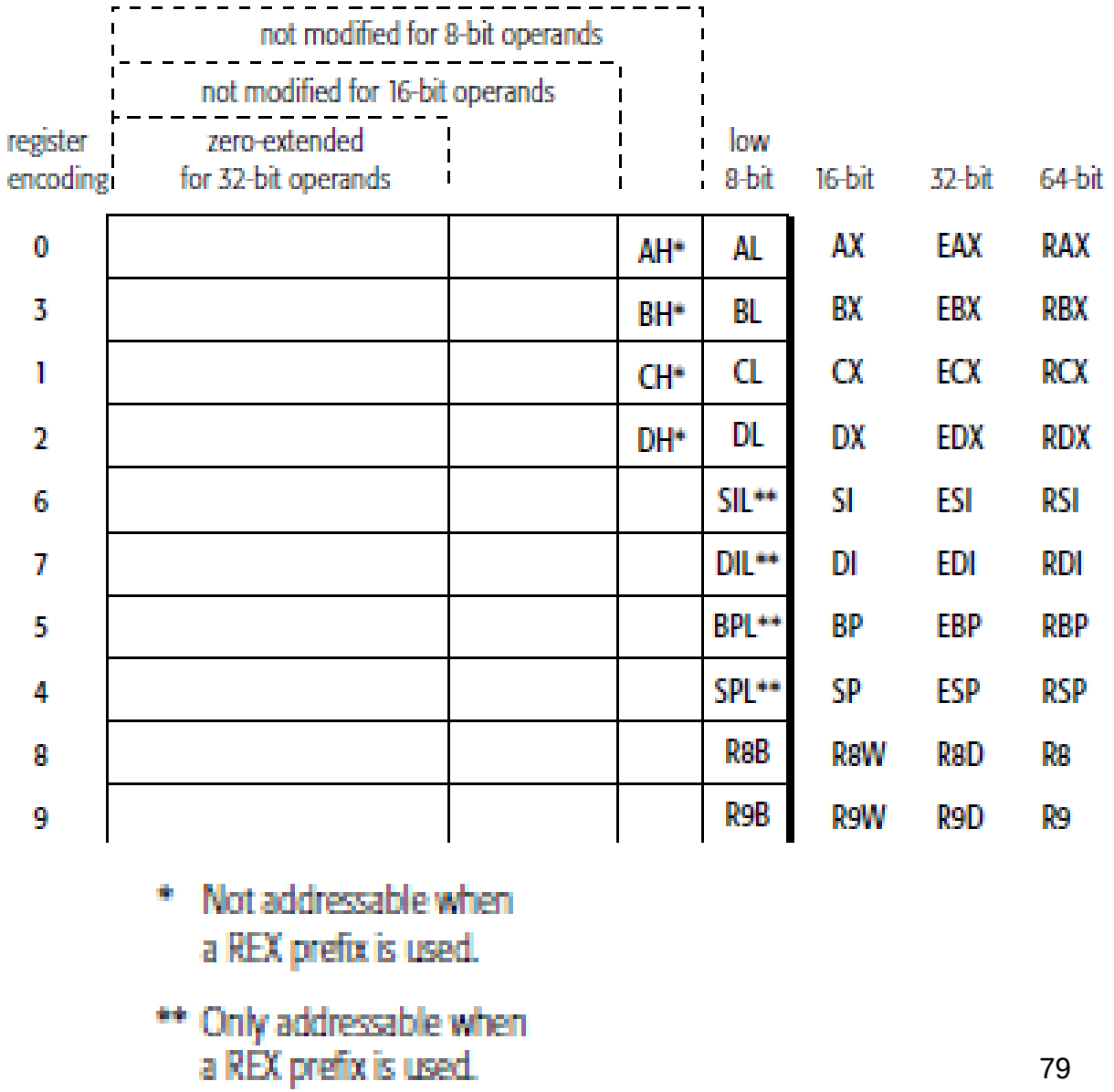
Nota

- Nel repertorio originale/iniziale 8086 erano previsti due codici per ADD
  - 00C3 per addizionare 8 bit (lasciando invariati gli altri 8)
  - 01C3 per addizionare 16
- Nel passare a 32 bit i due codici rimasero invariati
  - La somma a 16 bit invariata (solo sui 16 bit meno significati, lasciando invariati o 16 più significativi)
  - Per sommare 32 bit si mantenne il codice 01C3, aggiungendo un “operand-size prefix”
- Nel passare a 64 bit i codici sono rimasti invariati
  - Per sommare 64 bit è stato mantenuto il codice 01C3 ed è stato introdotto il prefisso REX
  - Si può fare la somma dei 32 bit meno significativi, in tal caso i 32 più significativi vengono messi a zero

Prefisso REX

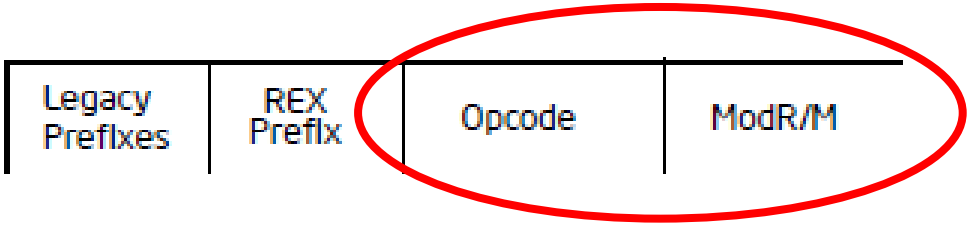
Come opera la somma in caso di operazioni su quantità a 32, 16 o 8 bit

Notare che le op su 32 bit si comportano in modo “non legacy”
(vedere più avanti)



Esempio (effetto REX)

- Mostriamo la somma di 2 registri, nella versione a 64, 32, 16, 8 bit
  - 64: ADD RBX,RAX
  - 32: ADD EBX,EA
  - 16: ADD BX,AX
  - 8: ADD BL,AL



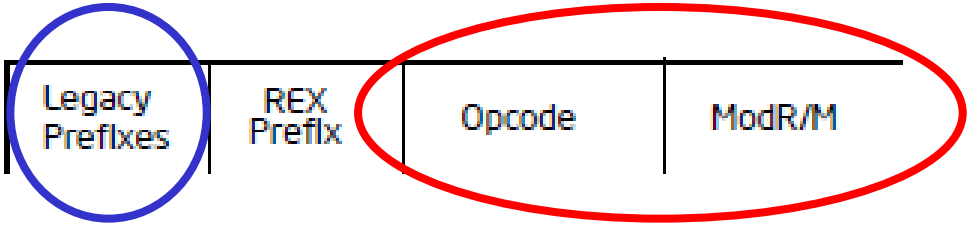
Codifica ADD (repertorio x32), pari a:

- 00C3 per 8 bit
- 01C3 per 16, 32 e 64 bit

Richiede il prefisso legacy operand-size override per scegliere

Esempio (effetto REX)

- Mostriamo la somma di 2 registri, nella versione a 64, 32, 16 bit
  - 64: ADD RBX,RAX
  - 32: ADD EBX,EA
  - 16: ADD BX,AX
  - 8: ADD BL,AL

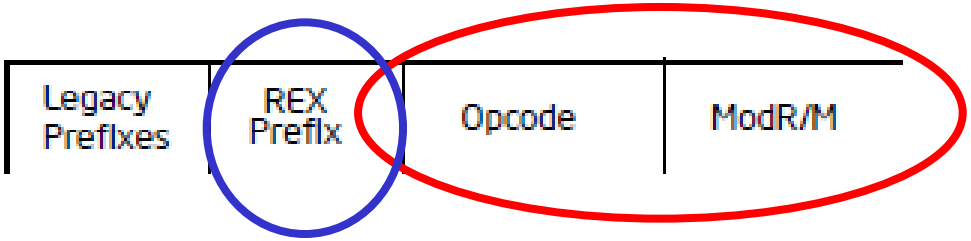


- 01C3 per 16, 32 e 64 bit

Richiede il prefisso legacy operand-size override per scegliere: vale **66**

Esempio (effetto REX)

- Mostriamo la somma di 2 registri, nella versione a 64, 32, 16 bit
  - 64: ADD RBX,RAX
  - 32: ADD EBX,EA
  - 16: ADD BX,AX
  - 8: ADD BL,AL



- 01C3 per 16, 32 e 64 bit

Richiede il prefisso REX per scegliere: **vale 48**

Esempio: caso 1

64-bit Add:

- Contenuto registri prima dell'operazione:
  - RAX =0002_0001_8000_2201
  - RBX =0002_0002_0123_3301

- Codifica ADD RBX,RAX: 48 01C3
  - 48 è il prefisso REX dimensione operandi a 64 bit.
- Risultato
  - RBX = 0004_0003_8123_5502

Esempio: caso 2

32-bit Add:

- Contenuto registri prima dell'operazione:
  - RAX =0002_0001_8000_2201
  - RBX =0002_0002_0123_3301

- Codifica ADD EBX,EAX: 01C3
  - Nessun prefisso: operazione a 32 bit
- Risultato
  - RBX = 0000_0000_8123_5502
    - Il risultato a 32 bit è esteso con zeri a 64
      - E' questo il comportamento "non legacy"

Esempio: caso 3

16-bit Add:

- Contenuto registri prima dell'operazione:
  - RAX =0002_0001_8000_2201
  - RBX =0002_0002_0123_3301

- Codifica ADD BX,AX: 66 01C3
  - 66 è il prefisso (legacy) di override per avere operandi a 16 bit
- Risultato
  - RBX = 0002_0002_0123_5502
    - I bit 63-16 non sono stati toccati

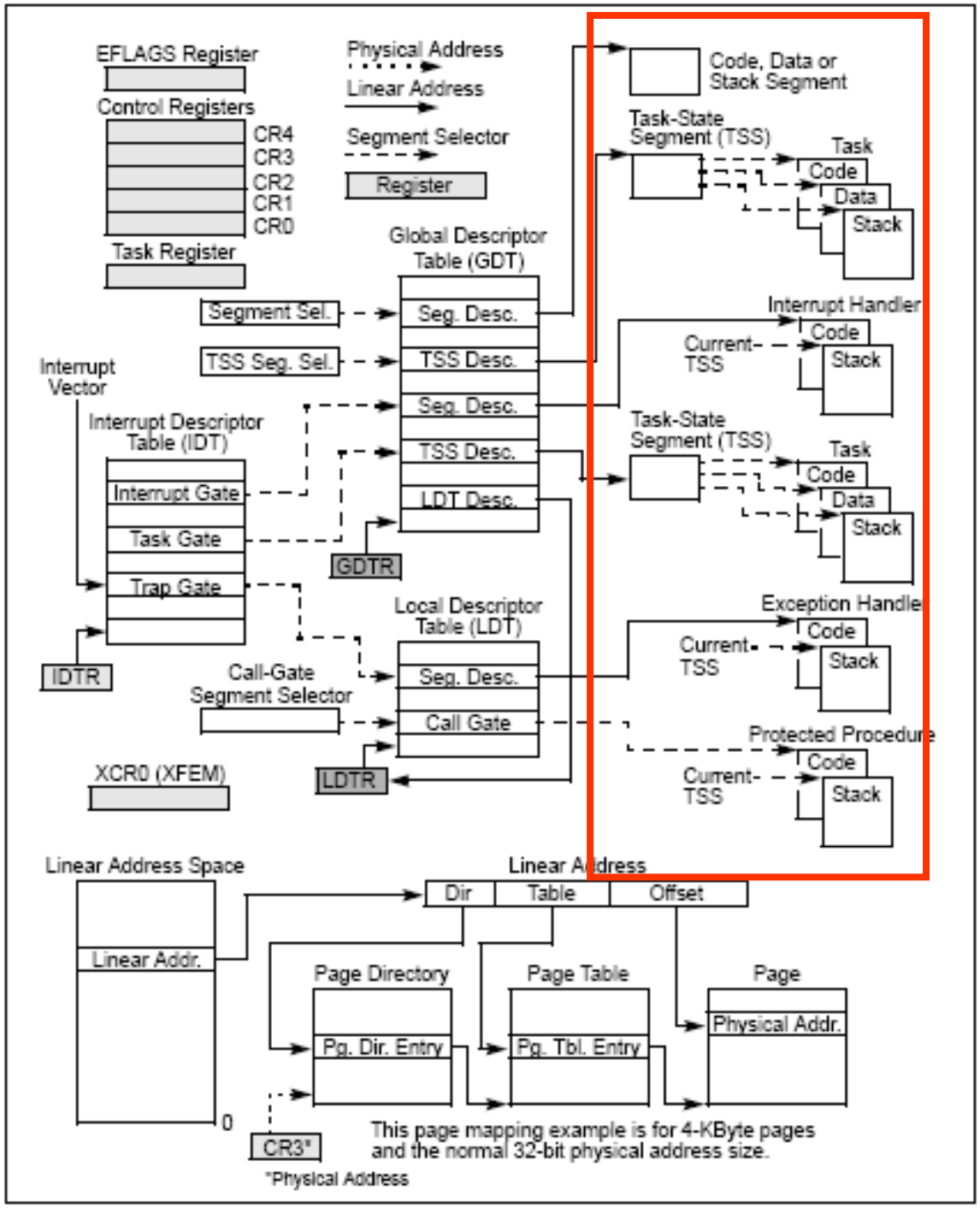
Esempio: caso 4

8-bit Add:

- Contenuto registri prima dell'operazione:
  - RAX =0002_0001_8000_2201
  - RBX =0002_0002_0123_3301

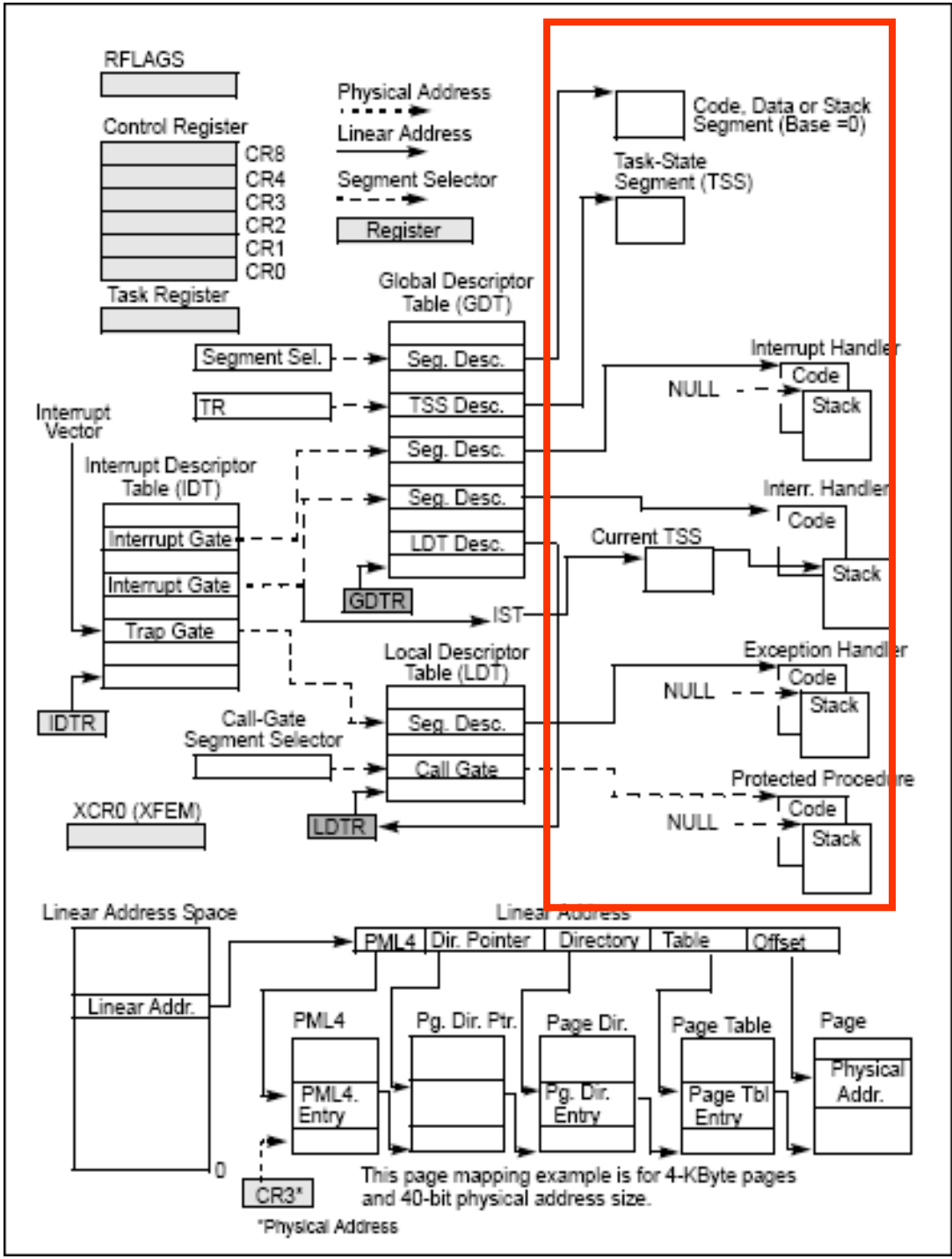
- Codifica ADD BL,AL: 00C3
  - Nessun prefisso: operazione a 8 bit, codice esplicito nel repertorio
- Risultato
  - RBX = 0002_0002_0123_3302
    - I bit 63-8 non sono stati toccati

IA-32

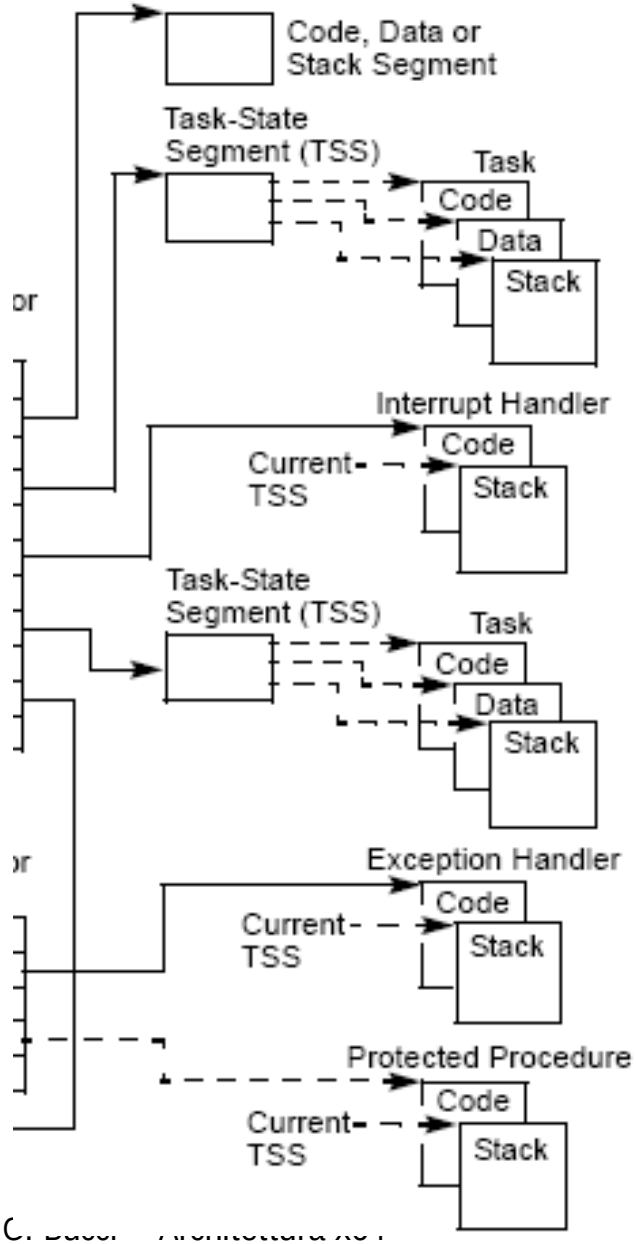

EFLAGS Register
Control Registers
CR4
CR3
CR2
CR1
CR0
Task Register
Physical Address
Linear Address
Segment Selector
Register
Global Descriptor Table (GDT)
Segment Sel.
TSS Seg. Sel.
Seg. Desc.
TSS Desc.
Seg. Desc.
TSS Desc.
LDT Desc.
Interrupt Vector
Interrupt Descriptor Table (IDT)
Interrupt Gate
Task Gate
Trap Gate
GDTR
IDTR
Local Descriptor Table (LDT)
Call-Gate Segment Selector
Seg. Desc.
Call Gate
LDTR
XCR0 (XFEM)
Code, Data or Stack Segment
Task-State Segment (TSS)
Task
Code
Data
Stack
Interrupt Handler
Current-TSS
Exception Handler
Protected Procedure
Linear Address Space
Linear Addr.
Linear Address
Dir
Table
Offset
Page Directory
Page Table
Page
Pg. Dir. Entry
Pg. Tbl. Entry
Physical Addr.
0
CR3*
This page mapping example is for 4-KByte pages and the normal 32-bit physical address size.
*Physical Address

IA-64

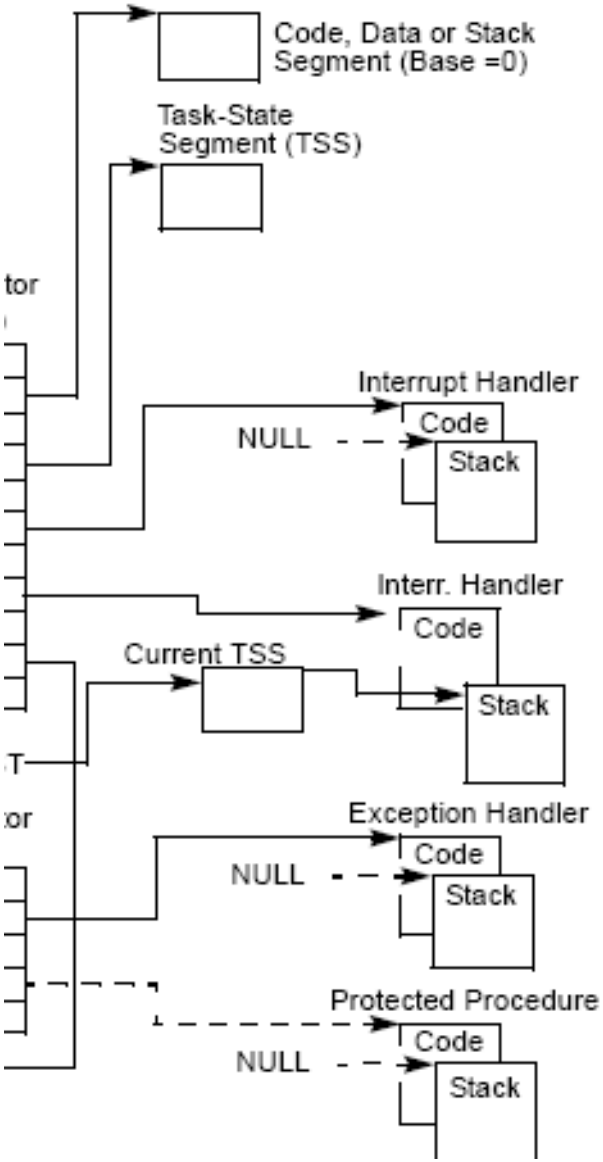
- A parte la MMU le differenze sono:
  - è sparito il Task Gate
  - non c'è più il task switching hardware (mantenuto nel modo IA-32e)
  - il TSS contiene (solo) i puntatori agli stack dei differenti livello di privilegio



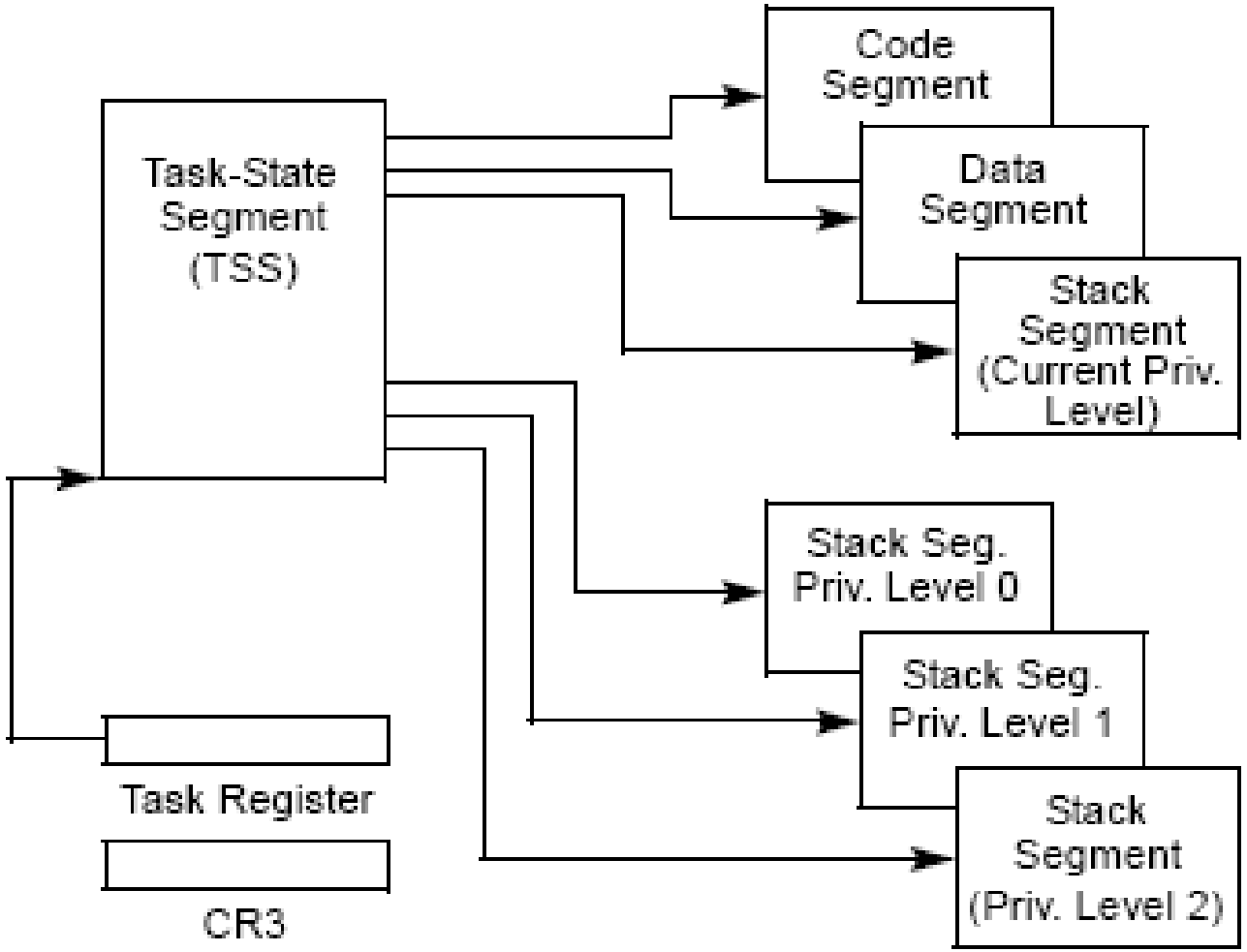
IA-32


Code, Data or
Stack Segment
Task-State
Segment (TSS)
Task
Code
Data
Stack
Interrupt Handler
Code
Current
TSS
Stack
Task-State
Segment (TSS)
Task
Code
Data
Stack
Exception Handler
Code
Current
TSS
Stack
Protected Procedure
Code
Current
TSS
Stack

IA-64


Code, Data or Stack
Segment (Base =0)
Task-State
Segment (TSS)
Interrupt Handler
Code
NULL
Stack
Interr. Handler
Code
Current TSS
Stack
Exception Handler
Code
NULL
Stack
Protected Procedure
Code
NULL
Stack

Struttura task a 32 bit



Task switching x32

- La logica passa a un altro task quando:
  - Il task corrente (o programma o procedura) esegue una JMP o una CALL a
    - un descrittore di TSS in GDT
    - a un task-gate in GDT or nella LDT corrente.
  - Un'interruzione o eccezione punta a un task-gate in IDT
  - Il task corrente esegue una IRET quando il bit NT di EFLAGS è asserito
- Il processo è piuttosto complesso e porta a cambiare il contesto della CPU in base al contenuto nel TSS del task entrante. Vengono eseguiti controlli di privilegio ecc..

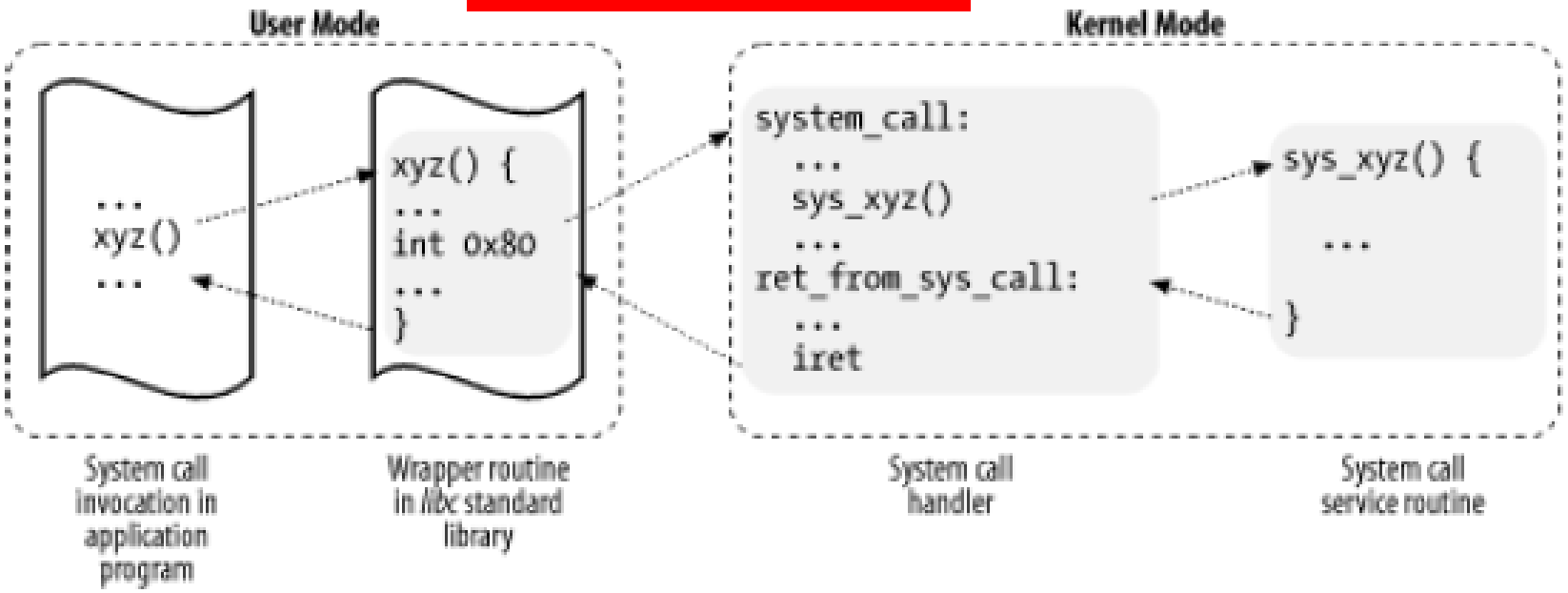
Task switching x64

- Non supportato. La gestione dei processi deve essere necessariamente SW
  - La logica genera un'eccezione se si tenta di trasferire il controllo a un TSS o task-gate attraverso JMP, CALL, INTn o Interruzione esterna o IRET (il sistema operativo interviene ed effettua il task switching)
- Il TSS assume un altro formato
  - Finisce per contenere solo puntatori agli stack dei diversi privilegi e puntatori di interruzione (oltre ai privilegi di I/O)
  - Un solo TSS basta per tutti
- E' un adeguamento a quel che fanno i sistemi operativi anche a 32 bit (Linux, Windows)

Esempio: SysCall di Linux

- Fino alla versione 2.5 le chiamate al kernel erano fatte con INT 0x80, usando EAX per identificare la specifica funzione

Proprio come il DOS!!!

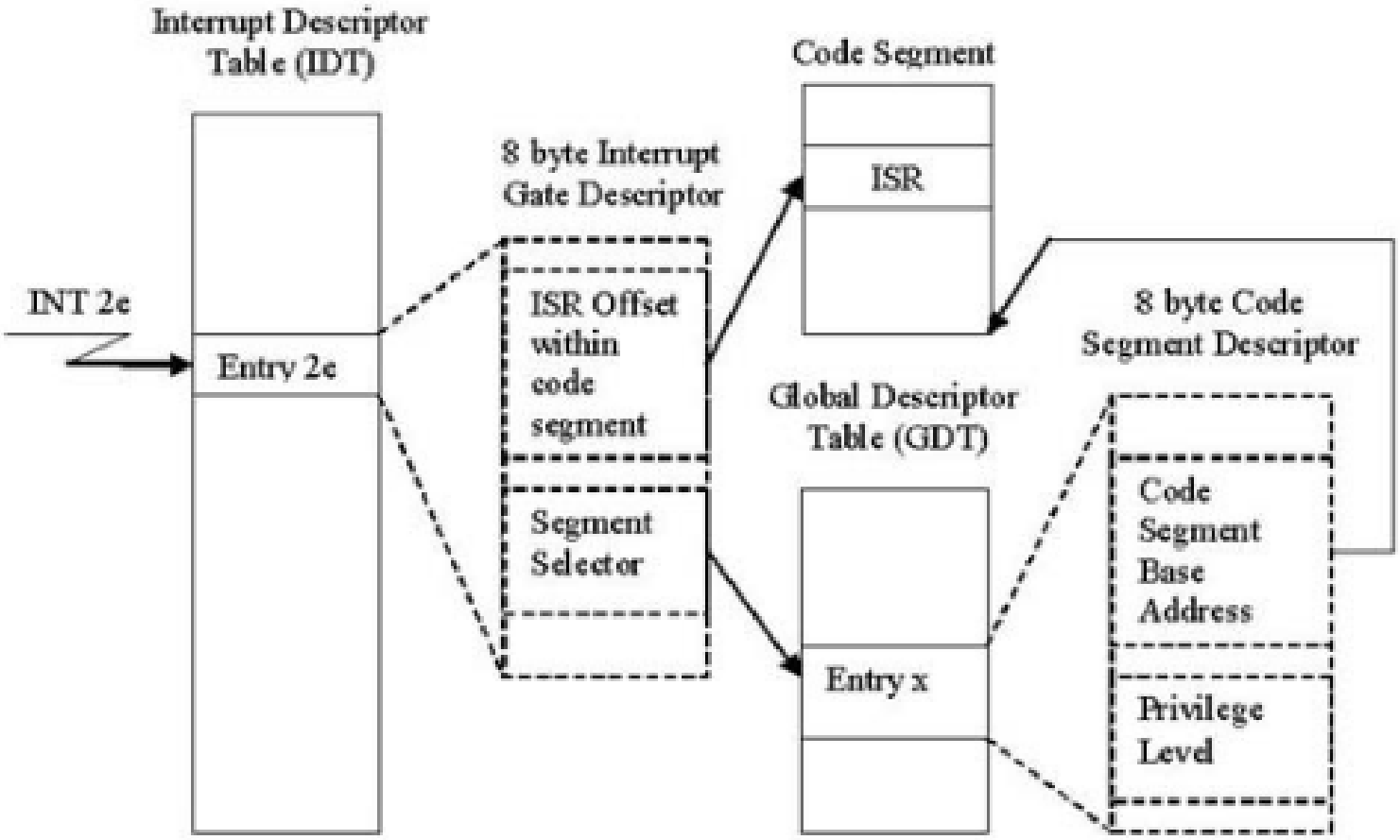


Analogamente Windows usa(va) INT 0x2e

Chiamate con INT

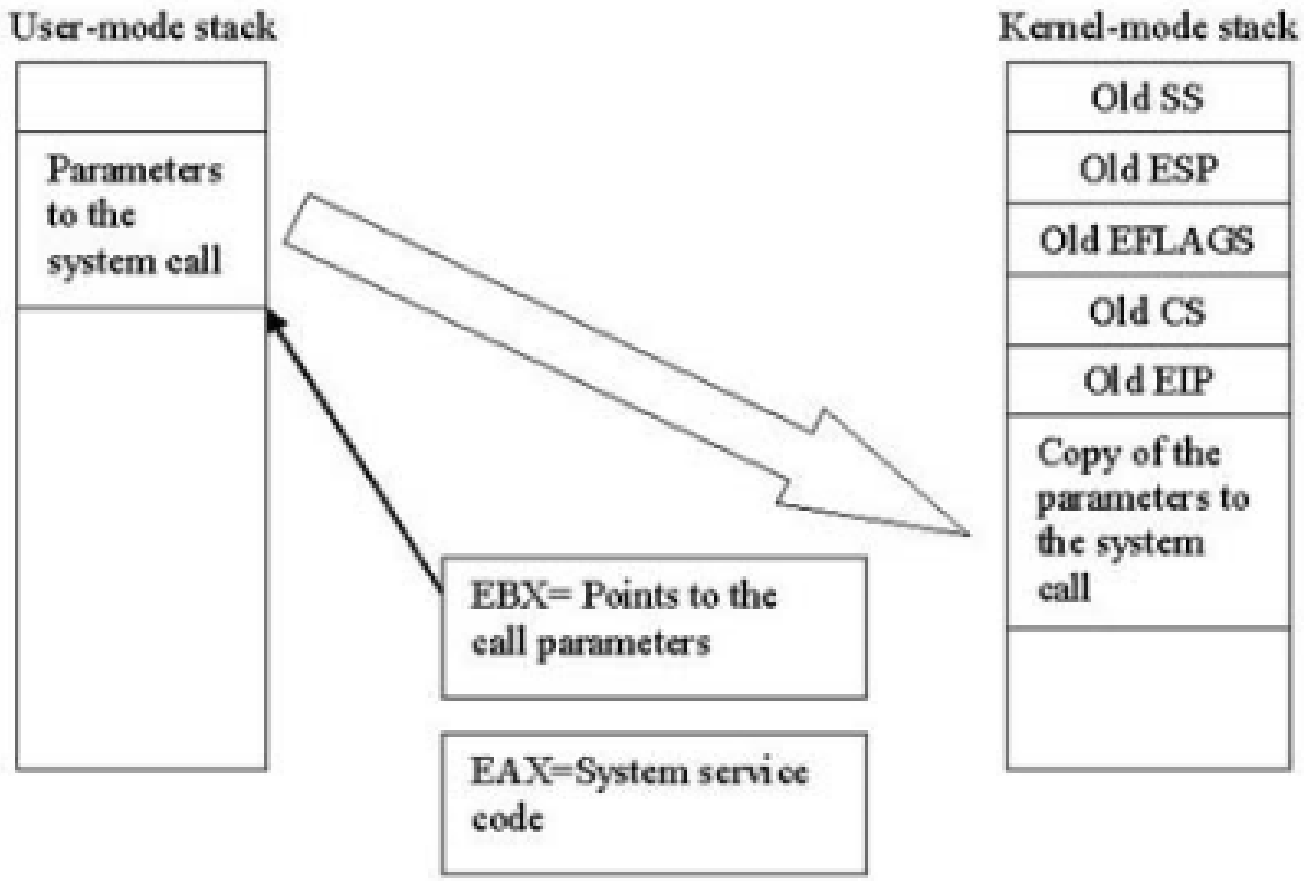
Windows

- Attraverso la porta di interruzione



Chiamata con INT: passaggio parametri

- Convenzioni (Windows)
  - La copiatura dei parametri la deve fare la funzione di SO



Finalmente: SYSENTER/SYSEXIT

- Introdotte col Pentium II per le chiamate al sistema operativo
- Motivate dal fatto che i sistemi operativi usano modelli flat e due soli livelli di protezione 0 e 3.
  - Kernel/SO Livello 0
  - User Livello 3
- Dette “fast system call/ret”, perché passano il controllo molto più velocemente delle usuali chiamate o interruzioni, poiché eliminano i controlli di privilegio ed effettuano il passaggio direttamente
- Prendono le “*destinazioni”* attraverso “registri” (locazioni di memoria) predefiniti
- Non salvano l’indirizzo di ritorno; il passaggio di eventuali parametri deve essere effettuato stabilendo una convenzione

SYSENTER

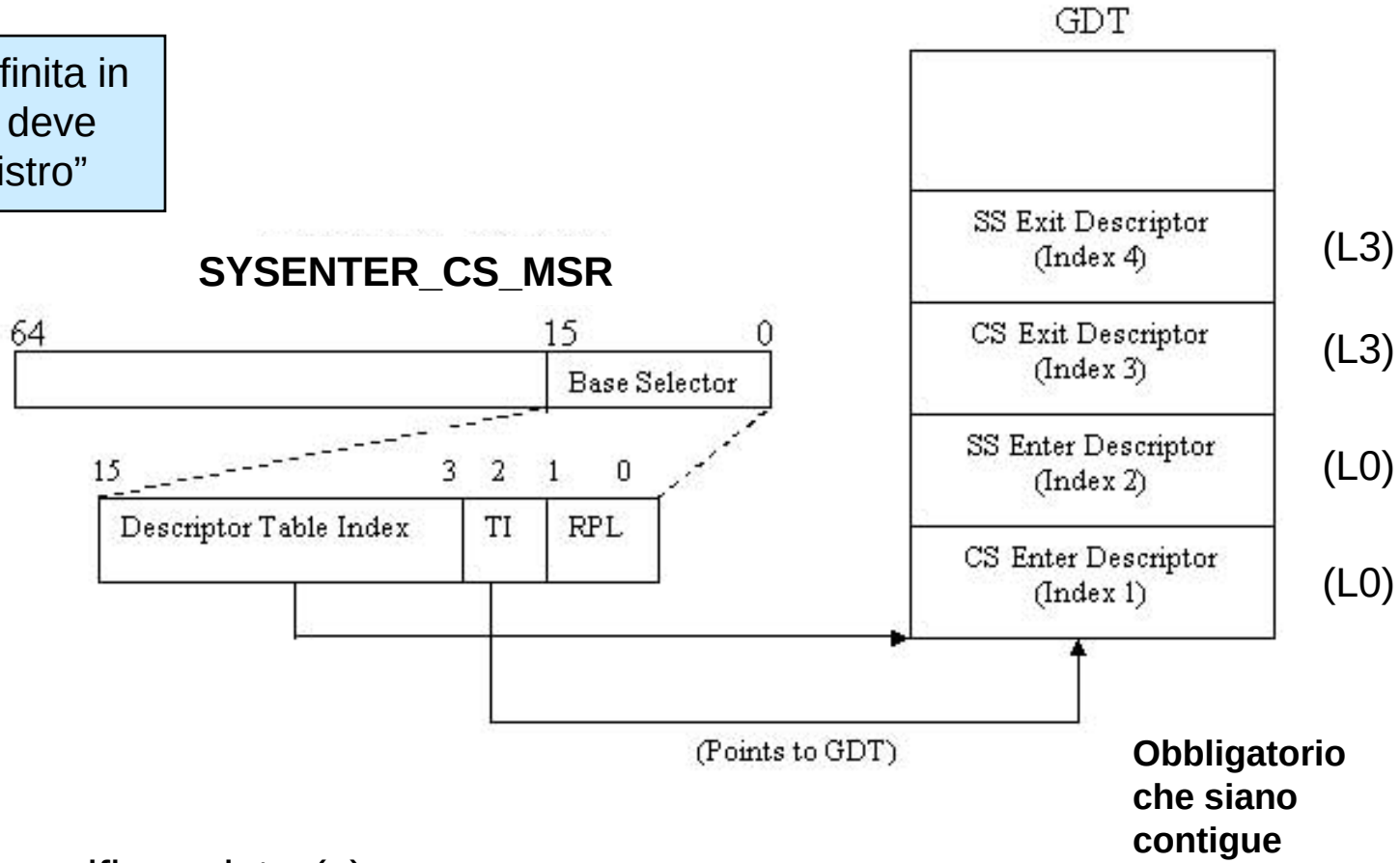
- Usata da codice a livello 3 (user) per passare a codice a livello 0 (kernel)
- Per convenzione la chiamata usa questi tre "*registri*"
  - SYSENTER_CS_MSR deve contenere il selettore del segmento di codice di livello 0
  - SYSENTER_EIP_MSR deve contenere l'offset nel segmento di codice di levello 0
  - SYSENTER_ESP_MSR deve contenere il puntatore alla testa dello stack di livello 0
- Inoltre
  - Il descrittore dello stack di livello 0 deve seguire in GDT il descrittore del codice di livello 0
  - I due successivi descrittori in GDT devono essere quelli del codice di livello 3 e dello stack di livello 3

SYSENTER (x32)

Nota: non serve un Data segment !

Posizione predefinita in memoria a cui deve trovarsi il “registro”

174H



Obbligatorio che siano contigue

MSR: machine specific register(s)

(C’è un mucchio di questi registri nelle macchine correnti)

SYSEXIT (X32)

- E’ l’operazione che fa coppia con SYSENTER
- All’atto dell’esecuzione
  - I descrittori dei segmenti di codice e stack (del livello 3 cui tornare) sono in GDT cui si accede sempre attraverso SYSENTER_CS_MSR
  - EDX deve contenere l’offset cui saltare nel segmento di codice di livello 3 (viene copiato in EIP)
  - ECX deve contenere il puntatore alla testa dello stack di livello 3
- In modo X32e ci sono piccole differenze

SYSCALL/SYSRET

- Sono le SYSENTER/SYSEXIT per il modo 64 bit
- Possono essere usate solo in modo 64-bit
- Ovviamente ci sono differenze legate al fatto che qui il modello (hardware) è piatto, ma la loro funzione è quella di passare da livello 3 a livello 0 e viceversa

- **Conclusione**: Windows e Linux hanno sostanzialmente buttato alle ortiche i meccanismi di protezione dell'architettura X86

Sintesi

Mode		System Software Required	Application Recompile Required	Defaults[1]		Register Extensions[2]	Maximum GPR Width (bits)
				Address Size (bits)	Operand Size (bits)		
Long Mode[3]	64-Bit Mode	New 64-bit OS	yes	64	32	yes	64
	Compatibility Mode		no	32		no	32
				16	16		
Legacy Mode	Protected Mode	Legacy 32-bit OS	no	32	32	no	32
				16	16		
	Virtual-8086 Mode			16	16		32
	Real Mode	Legacy 16-bit OS					

Note:

1. *Defaults can be overridden in most modes using an instruction prefix or system control bit.*
2. *Register extensions includes eight new GPRs and eight new XMM registers (also called SSE registers).*
3. *Long mode supports only x86 protected mode. It does not support x86 real mode or virtual-8086 mode.*

- Sorgente: manuali AMD

Conclusioni

- L’architettura a 64 bit è ottenuta come specifica modalità operativa dell’architettura *legacy* a 32 bit
- E’ stata mantenuta la compatibilità SW
  - Programmi legacy possono operare invariati sotto un sistema operativo a 64 bit
- Nel funzionamento a 64 bit il modello di memoria è piatto, il task switching non è più supportato
  - Anni di esperienza hanno mostrato che i sistemi operativi usano il modello piatto e 2 soli anelli di protezione (0 per il SO e 3 per i programmi utente)
  - Lo stesso vale per quanto riguarda la gestione dei task