Sistemi Operativi

Introduzione all'architettura IA-32

Lez. 15

Microprocessori Intel

- Nel 1979 Intel introduce la famiglia dei microprocessore 8086
- 8086, 8087, 8088, e 80186
  - Processori a 16-bit con registri a 16-bit
  - Bus dati a 16-bit e bus indirizzi a 20-bit
  - 8087 monta un co-processore Floating-Point
  - 8088 è una versione economica dell' 8086
    - Usa un bus dati a 8-bit
  - 80186 è una versione più veloce dell' 8086

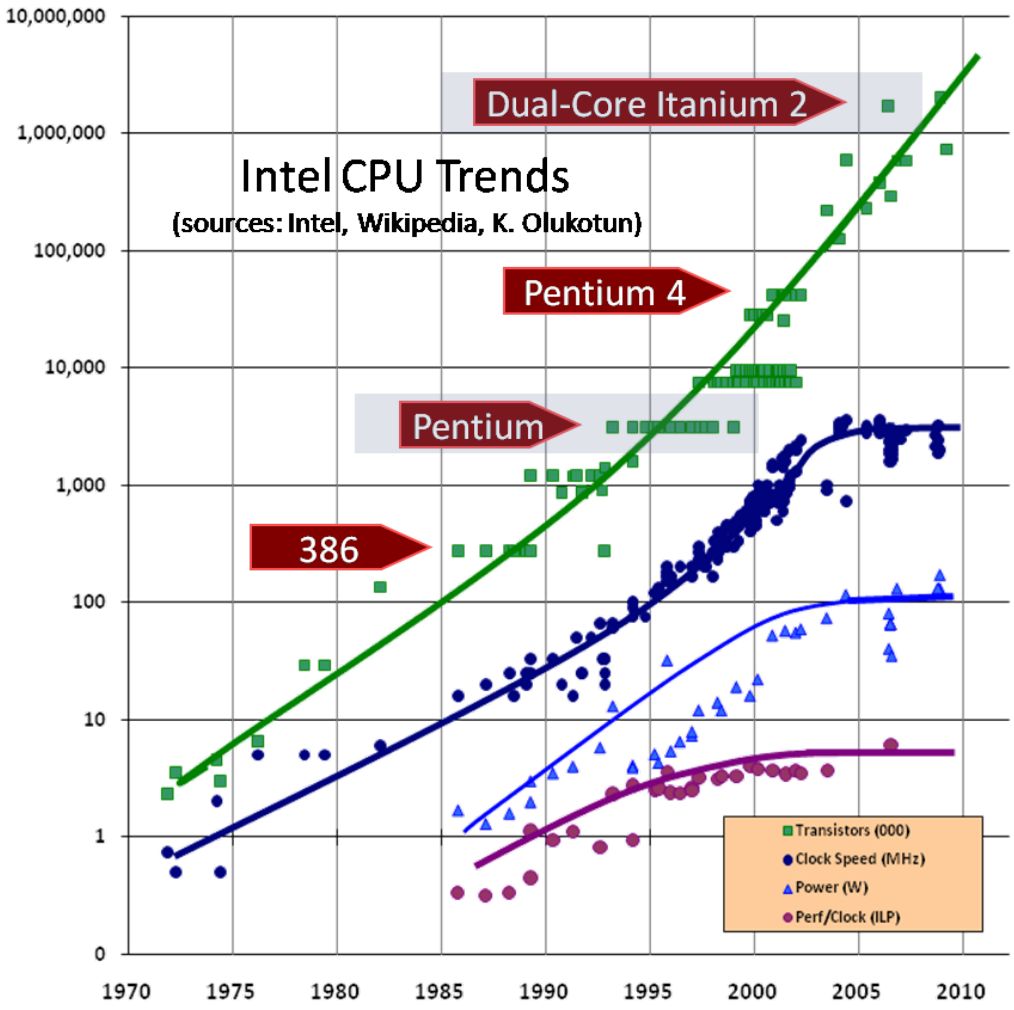
80286 e 80386

- 80286 introdotto nel 1982
  - Bus indirizzi a 24-bit $\Rightarrow 2^{24}$ bytes = 16 MB di indirizzamento
  - Introduce la modalità di gestione della memoria nota come protected mode
- 80386 introdotto nel 1985
  - Primo processore a 32-bit (32-bit general-purpose register)
  - 32-bit data bus e 32-bit address bus
  - 2^{32} bytes $\Rightarrow$ 4 GB di indirizzamento
  - Introduce nuove modalità di gestione della memoria: paging, virtual memory, e flat memory
    - La segmentazione può essere disabilitata

80486 e Pentium

- 80486 introdotto nel 1989
  - Versione migliorativa del 80386
  - On-chip Floating-Point unit
  - On-chip unified Instruction/Data Cache (8 KB)
  - Usa Pipelining: completa l'esecuzione di un'istruzione per ogni ciclo di clock
- Pentium (80586) introdotto nel 1993
  - 64-bit data bus, bus indirizzi resta a 32 bits
  - Due linee di pipeline: U-pipe e V-pipe
    - Superscalare: può completare l'esecuzione di 2 istruzioni per ciclo di clock
    - Cache dati e istruzioni separate di 8 KB ciascuna
  - Istruzioni MMX per applicazioni multimediali

Evoluzione



ARCHITETTURA IA-32

Modalità operative

- L'architettura IA-32 (a partire da Intel386) fornisce una serie di funzionalità per la realizzazione di software di sistema, queste funzionalità variano in funzione delle modalità con cui opera il processore :
  - Real mode
  - Protected mode
  - Virtual 8086 mode
  - System Management mode
  - IA-32e mode

Modalità operative

- Real-address mode: è la modalità operativa in cui il processore fornisce le stesse funzionalità presenti nell'Intel 8086 con l'aggiunta di alcune estensioni (la possibilità di cambiare modalità operativa in protected o system management mode)
- Protected mode: la modalità nativa del processore che fornisce un insieme di funzionalità predefinite per la realizzazione di sistemi multiprogrammati garantendo back compatibilità
- Virtual-8086 mode : dalla modalità protected è possibile passare alla modalità virtual-8086 mode, che consente al processore di eseguire software predisposto per l'architettura 8086 in un ambiente multiprogrammato e protetto

Modalità operative

- System management mode (SMM): SMM introdotto nel 1990 sul processore 386SL, è una modalità che consente al processore di eseguire codice memorizzato in una zona riservata della memoria nota come SMRAM (System Management RAM). Il codice memorizzato in questa porzione di memoria viene quindi eseguito ad livello di priorità estremamente elevato (-2)
  - Usato per la gestione di funzionalità hardware: accensione spegnimento dispositivi, diagnostica di componenti

IA-64

- Con l'avvento delle architetture a 64 bit è stata introdotta la nuova modalità operativa IA-32e che consente sia il supporto di applicazioni a 32 bit che quello di applicazioni a 64, più precisamente il software può essere eseguito in due sotto modalità:
  - 64-bit mode per il supporto di applicazioni e OS a 64-bit
  - Compatibility mode: che consente ad un sistema operativo a 64 bit di gestire applicazioni a 64/32-bit

Modalità processore

- All'accensione il processore viene automaticamente predisposto in real-address mode. Il valore del flag PE nel registro di controllo CR0 determinerà successivamente se il processore stia operando in real o protected mode
- Il processore entra in modalità SMM ogni volta che riceve un interrupt SMI, eseguito in una delle modalità real-address, protected, virtual-8086, o IA-32e modes.
- Il processore rientra nella sua modalità "normale" a seguito dell'esecuzione di un'istruzione RSM

Modalità processore

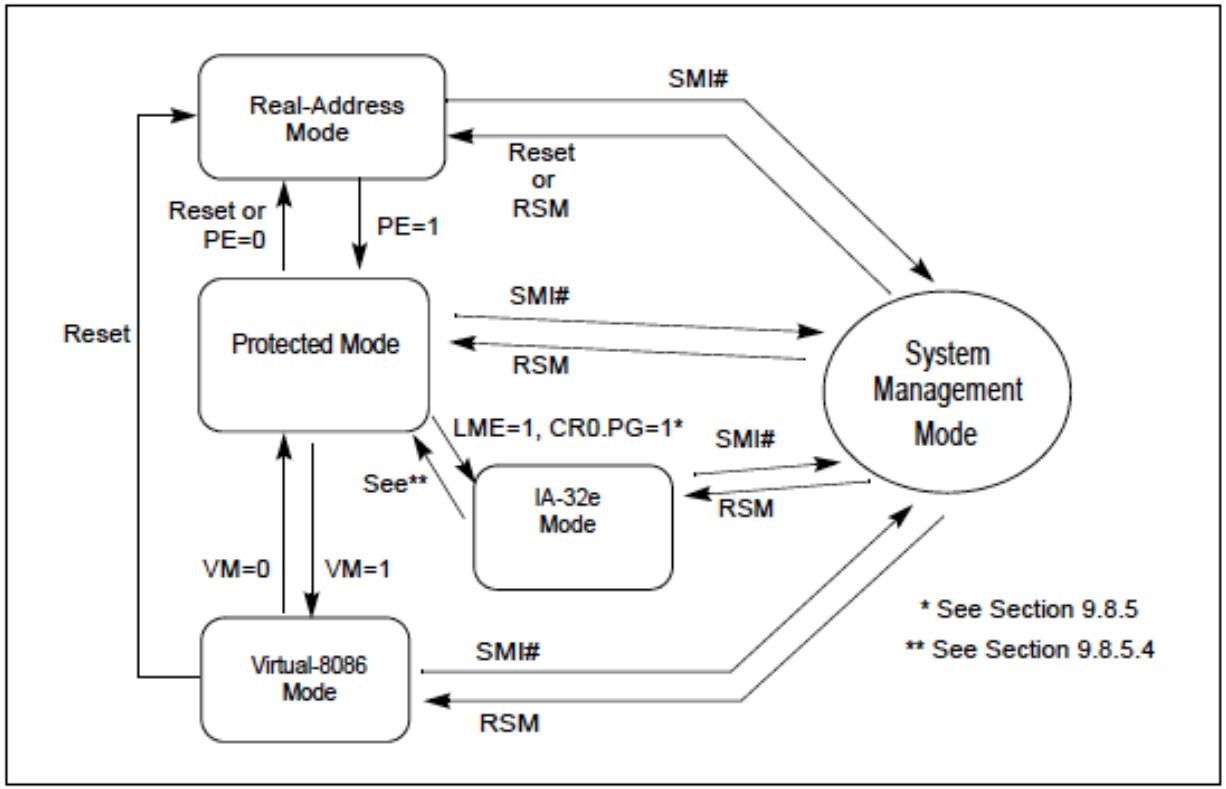


Figure 2-3. Transitions Among the Processor's Operating Modes

Protected mode

- Modalità introdotta per facilitare la soluzione di problemi di "sicurezza" derivanti dal ricorso al paradigma della multiprogrammazione. A tal fine l'architettura IA-32 include meccanismi di protezione e separazione che si prefiggono i seguenti obiettivi:
  - impedire a un processo di modificare l'area dati di un altro processo (restrizione sulla visibilità della memoria)
  - impedire a un processo di accedere direttamente alle risorse del sistema operativo
  - in caso di errore in un task, assicurare la sopravvivenza del sistema operativo e non compromettere la funzionalità degli altri task

Protected Mode

- I meccanismi predisposti dall'hardware per risolvere i suddetti problemi sono basati su due principi:
  - ISOLAMENTO: ad ogni processo utente è assegnato uno spazio di indirizzamento diverso, quindi processi diversi non hanno modo di interferire tra loro, se non attraverso sistemi controllati di IPC
  - La porzione di memoria dedicata al SO non può essere acceduta da processi utente
  - Protezione: ai diversi oggetti che popolano il sistema sono assegnati livelli di protezione (0-1-2-3), oggetti con livello di protezione k possono accedere solo ad oggetti con un livello di protezione uguale o maggiore a k

I Segmenti

- Il meccanismo di isolamento è implementato attraverso il processo di segmentazione, i segmenti sono porzioni di memoria DISGIUNTE in cui sono caricati processi, dati e opportune strutture dati
- Nella modalità di esecuzione protected i processi utente devono essere “spezzati” in due o più segmenti che possono essere di due tipi, in funzione del contenuto: codice o Dati/ stack
- I processi sono gestiti dal processore attraverso un apposito segmento noto come Task State Segment (TSS)
- Un ulteriore segmento (opzionale) è costituito da opportune tabelle, con l’elenco degli oggetti accessibili a un processo (LocalDescriptorTable LDT)

I segmenti

- Le tipologie di segmento imposte dall'architettura e che studieremo sono quindi:
  - Codice
  - Dati/stack
  - TSS
  - LDT

Descrittori

- Ciascuno dei segmenti sopra indicati deve essere provvisto di un opportuna struttura dati che ne descrive le principali proprietà, questa struttura dati è chiamata DESCRITTORE
- Un descrittore di segmento fornisce al processore le seguenti informazioni:
  - La dimensione e l'indirizzo di partenza di un segmento,
  - Informazioni per il controllo degli accessi e sullo stato del segmento
- I descrittori di segmento sono generati dal SO o da programmi di sistema

Contenuti di un Segment Descriptor

- Base Address
  - Un numero di 32-bit che definisce l'indirizzo di partenza del segmento
  - 32-bit Base Address + 32-bit Offset = 32-bit Linear Address
- Segment Limit
  - Un numero di 20-bit che specifica la dimensione del segmento
  - La dimensione può essere specificata in byte o pagine (4 KB)
- Diritti di accesso
  - Se il segmento contiene codice o dati
  - Se i dati sono read-only, in sola lettura o anche scrivibili
  - Il livello di privilegio del segmento

Descrittori speciali

- L'architettura prevede anche un insieme di descrittori speciali, non legati a segmenti, chiamati GATE (call gate, interrupt gate, trap gate e task gate)
- Si tratta di un meccanismo fornito dall'hw per controllare l'esecuzione di codice privilegiato da parte di applicazioni che operano in user space

Layout descrittore segmento (8 byte)

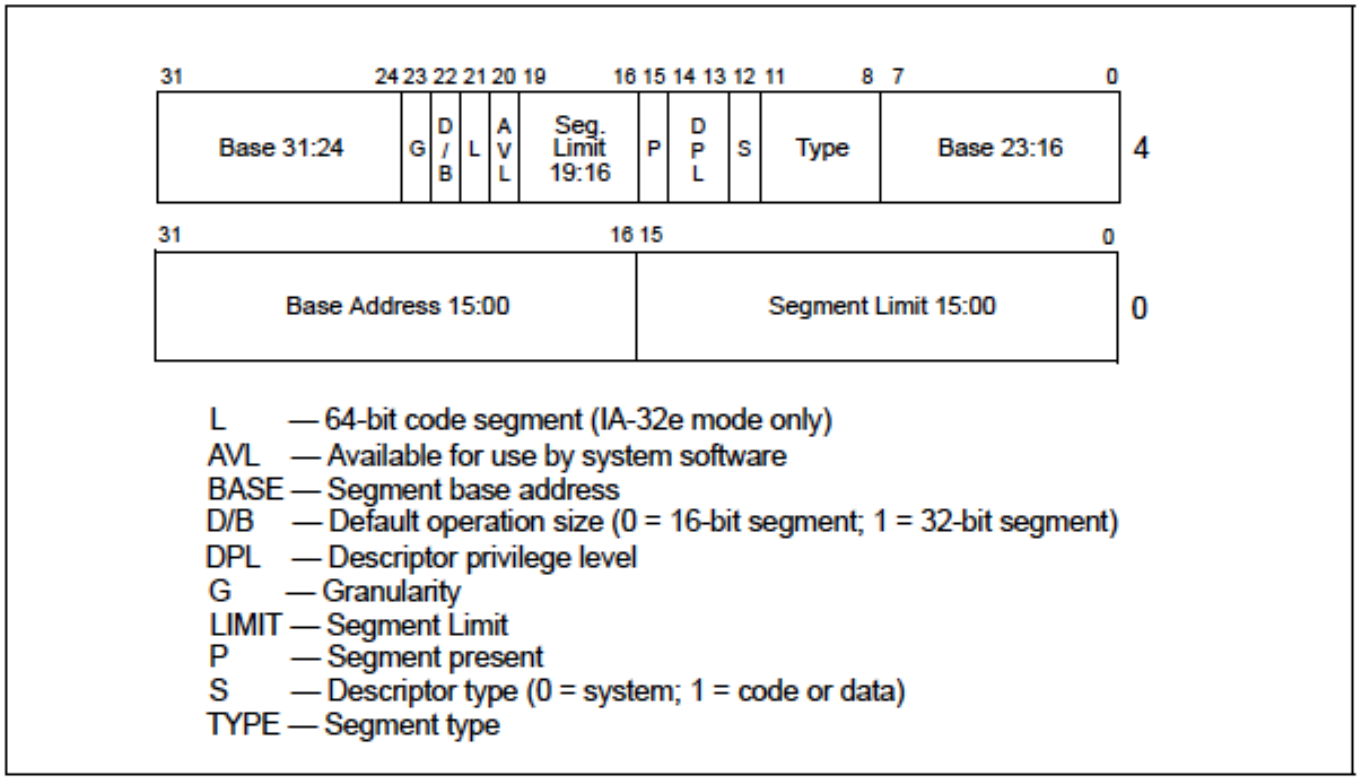


Figure 3-8. Segment Descriptor

Segment Descriptor

```
// Segment Descriptors
struct Segdesc {
    unsigned sd_lim_15_0 : 16;  // Low bits of segment limit
    unsigned sd_base_15_0 : 16; // Low bits of segment base address
    unsigned sd_base_23_16 : 8; // Middle bits of segment base address
    unsigned sd_type : 4;       // Segment type (see STS_ constants)
    unsigned sd_s : 1;          // 0 = system, 1 = application
    unsigned sd_dpl : 2;        // Descriptor Privilege Level
    unsigned sd_p : 1;          // Present
    unsigned sd_lim_19_16 : 4;  // High bits of segment limit
    unsigned sd_avl : 1;        // Unused (available for software use)
    unsigned sd_rsv1 : 1;       // Reserved
    unsigned sd_db : 1;         // 0 = 16-bit segment, 1 = 32-bit segment
    unsigned sd_g : 1;          // Granularity: limit scaled by 4K when set
    unsigned sd_base_31_24 : 8; // High bits of segment base address
};
```

Bit field

- ***Bit Fields*** allow the packing of data in a structure. This is especially useful when memory or data storage is at a premium. Typical examples
  - Packing several objects into a machine word. ***e.g.*** 1 bit flags can be compacted -- Symbol tables in compilers.
  - Reading external file formats -- non-standard file formats could be read in. ***E.g.*** 9 bit integers.
- C lets us do this in a structure definition by putting :***bit length*** after the variable. ***i.e.***

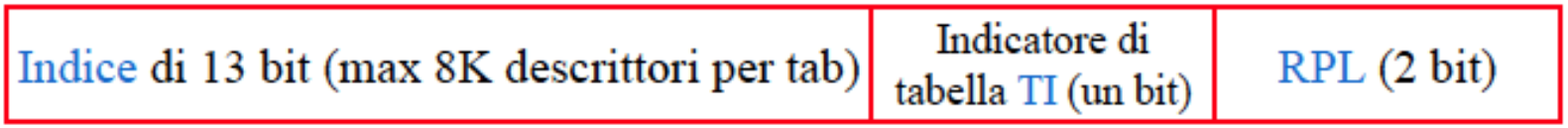
```
struct packed_struct {
      unsigned int f1:1;
      unsigned int f2:1;
      unsigned int f3:1;
      unsigned int f4:1;
      unsigned int type:4;
      unsigned int funny_int:9;
               } pack;
```

Tabelle dei descrittori

- A loro volta i DESCRITTORI sono radunati in TABELLE di DESCRITTORI
- Le TABELLE di DESCRITTORI sono organizzate in forma di vettori e sono referenziate attraverso REGISTRI di sistema
- Si accede ai DESCRITTORI all'interno delle TABELLE attraverso un indice lineare detto "SELETTORE"

Selettori

- I selettori hanno la seguente struttura:



- TI vale 1 nei selettori della LDT; TI vale 0 nei selettori della GDT
- RPL (Requestor Privilege Level) è il livello di previlegio del segmento di codice che ha generato il selettore

Le principali tabelle

- GDT o Global Descriptor Table è una tabella contente descrittori di segmenti accessibili a tutti i processi (non è un segmento, il suo accesso non avviene attraverso selettori)
- LDT o Local Descriptor Table è una tabella di descrittori di segmenti visibili solo dal processo (task) a cui la LDT è associata
- IDT o Interrupt Descriptor Table è una tabella contenente i descrittori dei Gates di accesso alle procedure o ai task associati agli interrupt. La IDT corrisponde alla Interrupt Table dell'8086. La IDT può contenere soltanto descrittori di Gate (Interrupt, Trap e Task Gate).

Costruzione GDT

```
# Bootstrap GDT
.p2align 2                                # force 4 byte alignment
gdt:
  SEG_NULL                # null seg
  SEG(STA_X|STA_R, 0x0, 0xffffffff)   # code seg
  SEG(STA_W, 0x0, 0xffffffff)          # data seg

gdtdesc:
  .word   0x17                                # sizeof(gdt) - 1
  .long   gdt                                 # address gdt
```

GDT

```
/*
 * Macros to build GDT entries in assembly.
 */
#define SEG_NULL                        \
    .word 0, 0;                         \
    .byte 0, 0, 0, 0
#define SEG(type,base,lim)                          \
    .word (((lim) >> 12) & 0xffff), ((base) & 0xffff); \
    .byte (((base) >> 16) & 0xff), (0x90 | (type)),         \
        (0xC0 | (((lim) >> 28) & 0xf)), (((base) >> 24) & 0xff)


// Application segment type bits
#define STA_X         0x8     // Executable segment
#define STA_W         0x2     // Writeable (non-executable segments)
#define STA_R         0x2     // Readable (executable segments)
#define STA_A         0x1     // Accessed
```

Registri di sistema

- Al fine di operare in modo efficiente sulle suddette tabelle la CPU IA-32 dispone dei seguenti registri:
  - GDTR o Global Descriptor Table Register, un registro inizializzato in sede di boot, che punta alla GDT della piattaforma
  - LDTR o Local Descriptor Table Register: un registro aggiornato ad ogni operazione di Task Switching, che punta alla LDT del Task Running
  - IDTR o Interrupt Descriptor Table Register: un registro inizializzato in fase di boot, che punta alla IDT della piattaforma
  - TR o Task Register: un registro che punta al TSS del Task Running

Registri di sistema

- Altri registri sono usati dal processore per controllare e gestire l'intero sistema, sono generalmente registri accessibili (almeno in parte) dal sistema operativo con istruzioni privilegiate
- I principali di questi registri sono:
  - EIP
  - EFLAGS
  - Control register (CR0, CR2, CR3, and CR4)
  - Segment Register

EFLAGS

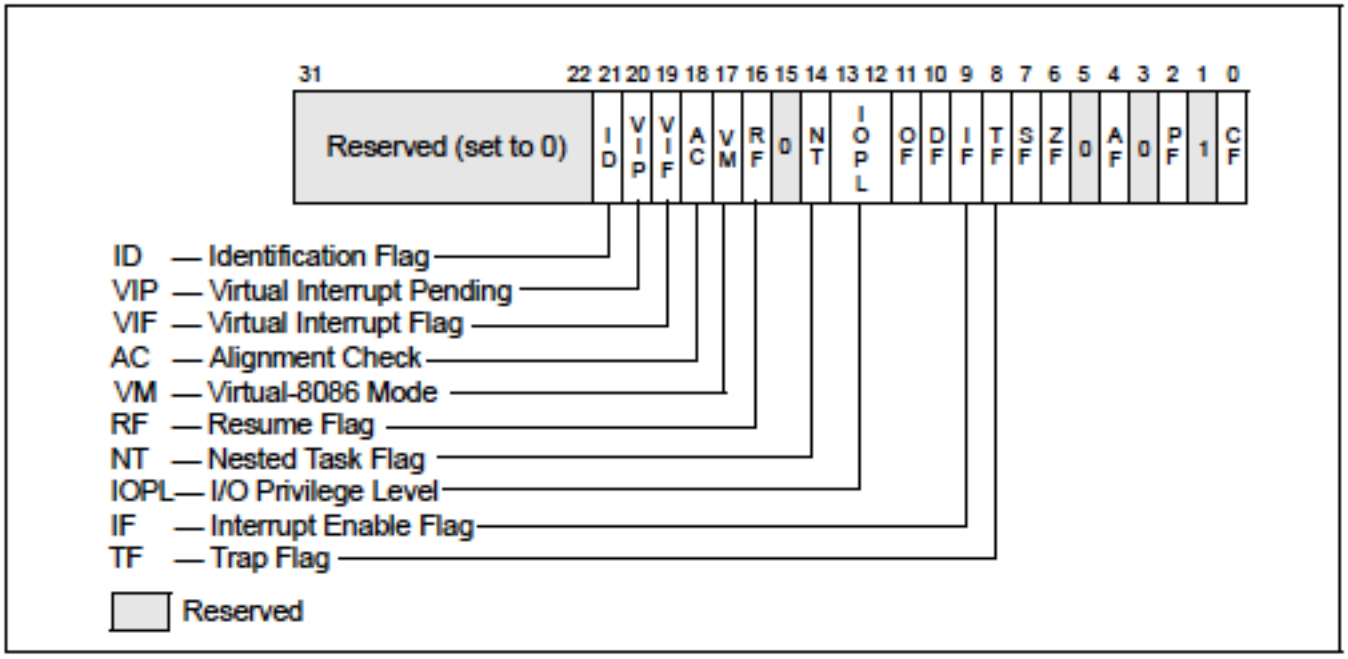


Figure 2-4. System Flags in the EFLAGS Register

Segment Register

- Sei 16-bit Segment Register
  - Servono per supportare la memoria segmentata, affinché un processo possa accedere ad un segmento è necessario che il selettore di quel segmento sia stato preventivamente caricato nel corrispondente registro
  - Sono usati come cache per i selettori di segmento
  - I Segmenti contengono diverse tipologie di dati
    - Codice
    - Dati
    - Stack

Segment Register

- Parte visibile = 16-bit Segment Register
  - CS, SS, DS, ES, FS, e GS sono visibili al programmatore
- Parte invisibile = Segment Descriptor (64 bits)
  - Caricato dall'hw

Visible part	Invisible part	
Segment selector	Segment base address, size, access rights, etc.	CS
Segment selector	Segment base address, size, access rights, etc.	SS
Segment selector	Segment base address, size, access rights, etc.	DS
Segment selector	Segment base address, size, access rights, etc.	ES
Segment selector	Segment base address, size, access rights, etc.	FS
Segment selector	Segment base address, size, access rights, etc.	GS

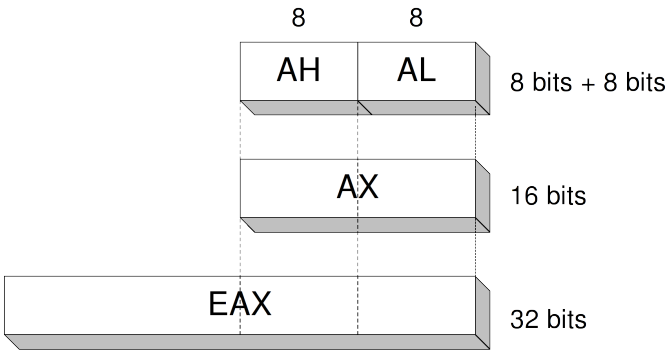
Registri General purpose

32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

32-bit	16-bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

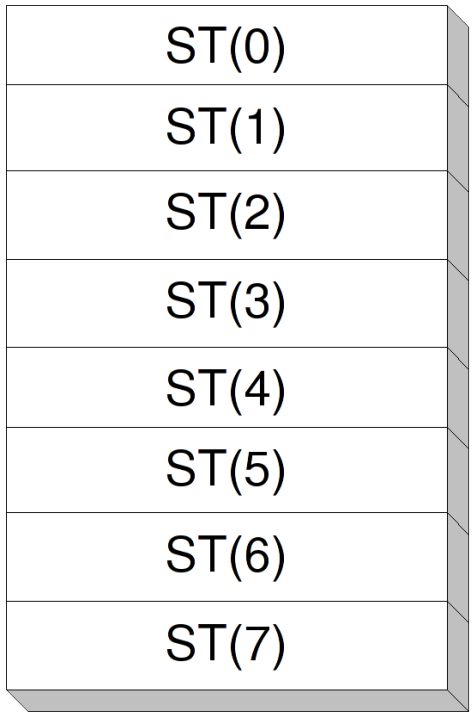
Accesso ai registri

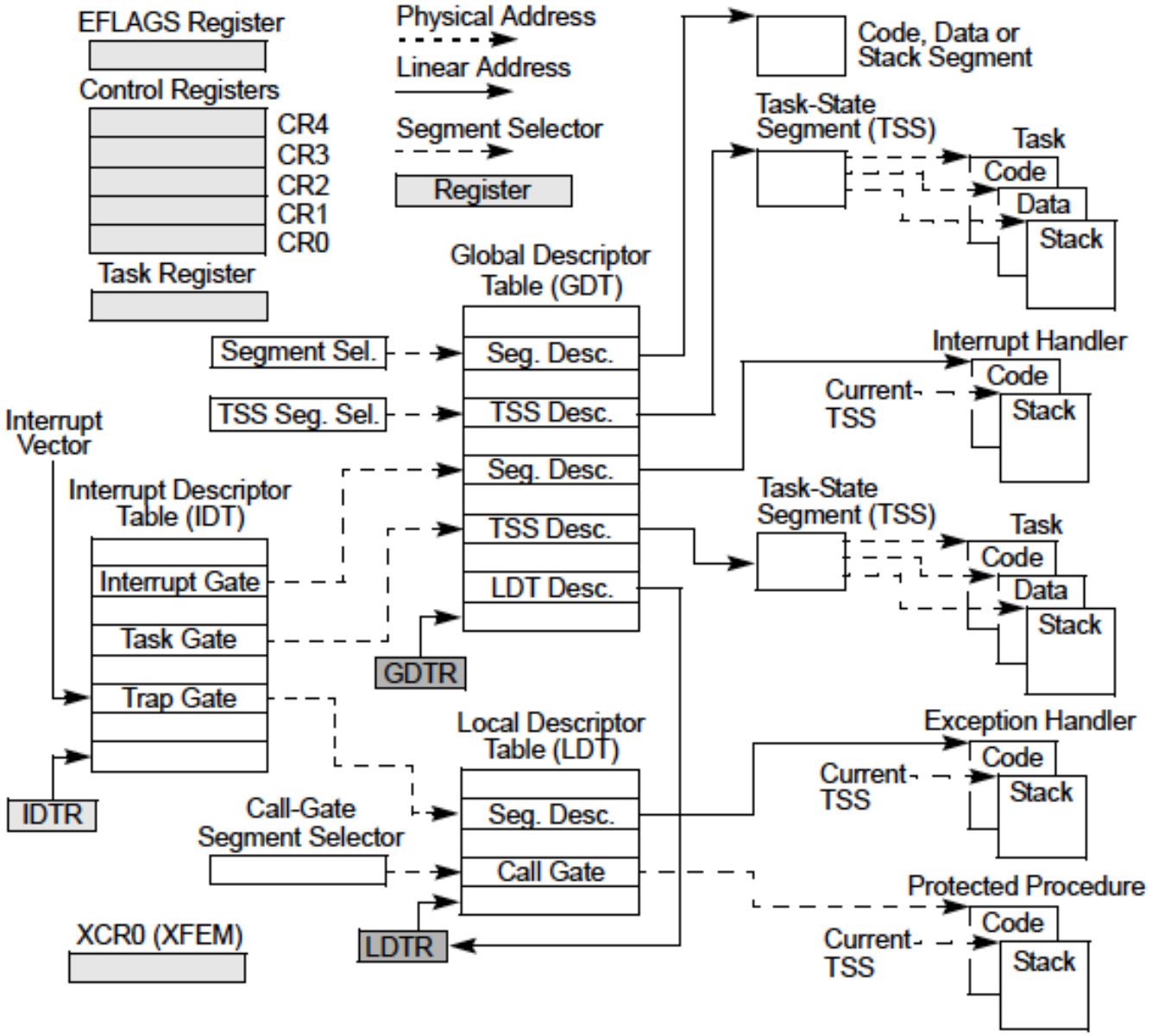
- EAX, EBX, ECX, e EDX sono registri a 32-bit
  - È possibile però accedere a soli 16-bit e 8-bit
  - I 16-bit meno significativi di EAX sono denotati con AX
  - AX è ulteriormente suddiviso
    - AL = 8 bit meno significativi
    - AH = 8 bit più significativi
- ESI, EDI, EBP, ESP: si può solo accedere ai 16 bit meno significativi



Registri Floating-Point, MMX, XMM

- Otto 80-bit floating-point data register
  - ST(0), ST(1), . . . , ST(7)
  - Usati da FPU
- Otto 64-bit MMX register
  - Usati dalle istruzioni MMX
- Otto 128-bit XMM registers
  - Usati dalle istruzioni SSE





Supporto ai processi

- Un task è definito come un'unità di lavoro che il processore può attivare, eseguire e sospendere. Può essere usato per eseguire un programma, una routine di sistema operativo, un gestore di interrupt ecc.
- Un task è composto da due componenti: uno spazio d'esecuzione e un task-state segment (TSS). Lo spazio d'esecuzione è costituito da un segmento codice, un segmento stack, e uno o più segmenti dati

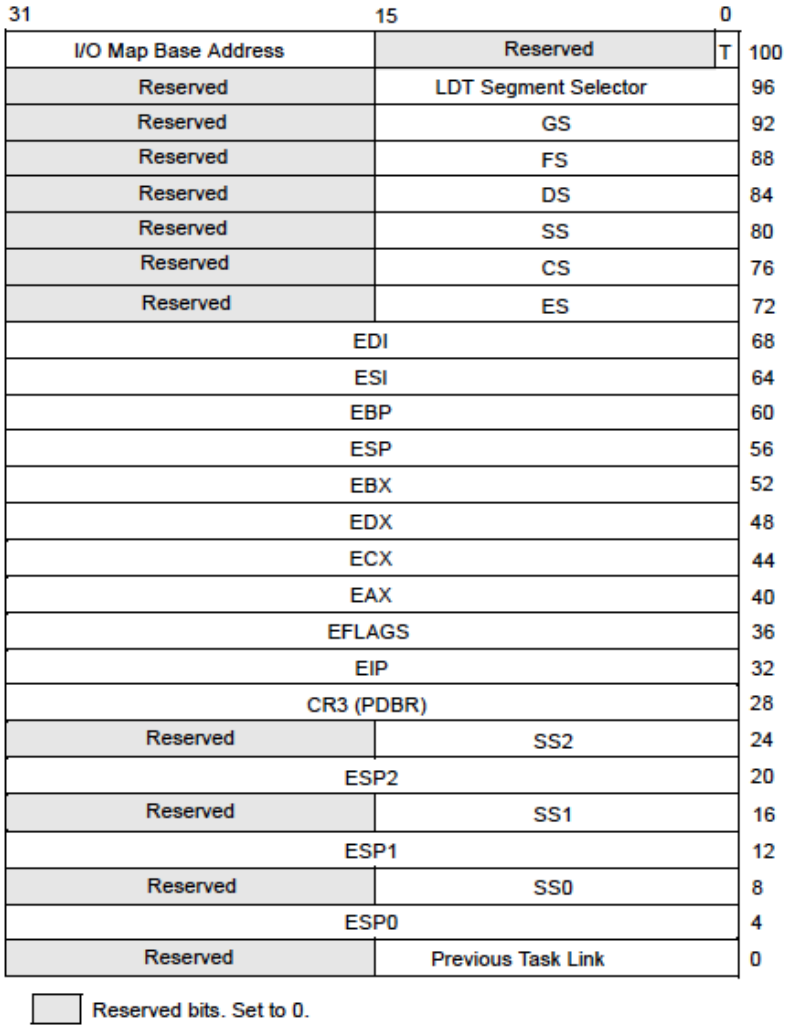
TSS

- Il TSS è un segmento che contiene un'opportuna struttura dati con le seguenti informazioni:
  - lo stato dell'ambiente d'esecuzione di un TASK,
  - i selettori e gli stack pointers a tre segmenti stack (uno stack per ogni livello di privilegio)
  - Il selettore di segmento per la LDT associata al task
  - Le informazioni necessarie per gestire la paginazione

Stato di un task

- Lo stato di un task è univocamente determinato:
  - Dal valore dei segment registers (CS, DS, SS, ES, FS, and GS)
  - Dal valore dei general-purpose register
  - Dal valore di EFLAGS
  - Dal valore di EIP
  - Dal valore del control register CR3
  - Dal valore del task register
  - Dal valore di LDTR
  - I valori per la gestione dell'I/O (contenuti in TSS)
  - Dagli Stack pointer 0, 1, e 2 (contenuti in TSS)

TSS



TSS

```
// Task state segment format (as described by the Pentium architecture book)
struct Taskstate {
        uint32_t ts_link;       // Old ts selector
        uintptr_t ts_esp0;      // Stack pointers and segment selectors
        uint16_t ts_ss0;        //   after an increase in privilege level
        uint16_t ts_padding1;
        uintptr_t ts_esp1;
        uint16_t ts_ss1;
        uint16_t ts_padding2;
        uintptr_t ts_esp2;
        uint16_t ts_ss2;
        uint16_t ts_padding3;
        physaddr_t ts_cr3;   // Page directory base
        uintptr_t ts_eip;       // Saved state from last task switch
        uint32_t ts_eflags;
        uint32_t ts_eax;        // More saved state (registers)
```

```
        uint32_t ts_ecx;
        uint32_t ts_edx;
        uint32_t ts_ebx;
        uintptr_t ts_esp;
        uintptr_t ts_ebp;
        uint32_t ts_esi;
        uint32_t ts_edi;
        uint16_t ts_es;             // Even more saved state (segment selectors)
        uint16_t ts_padding4;
        uint16_t ts_cs;
        uint16_t ts_padding5;
        uint16_t ts_ss;
        uint16_t ts_padding6;
        uint16_t ts_ds;
        uint16_t ts_padding7;
        uint16_t ts_fs;
        uint16_t ts_padding8;
        uint16_t ts_gs;
        uint16_t ts_padding9;
        uint16_t ts_ldt;
        uint16_t ts_padding10;
        uint16_t ts_t;          // Trap on task switch
        uint16_t ts_iomb;       // I/O map base address
};
```