

---

## Interrupt

La funzione del sistema I/O di un calcolatore è il trasferimento di informazioni tra la memoria principale ed i periferici.

I metodi usati dalla CPU per gestire le richieste di servizio da parte dei dispositivi sono di 3 tipi:

- Polling
- Interrupt
- DMA (trasferimenti a blocchi)

**Polling** La gestione polling viene fatta attraverso un ciclo software di lettura dello stato dei dispositivi I/O. Se il dispositivo interrogato ha richiesto un servizio, viene servito altrimenti si prosegue la scansione. La maggior parte del tempo è usata nell'esecuzione del ciclo di polling questo porta un'elevata latenza da parte del dispositivo.

**Interrupt** La cpu non interroga i dispositivi. Se un dispositivo ha bisogno di essere servito, questo fa richiesta alla cpu, la cpu vede se può interrompere quello che sta eseguendo, se sì, salva tutti i registri nello stack e passa all'esecuzione della routine di servizio associata a quella data richiesta di quel dato dispositivo. Il dispositivo che gestisce le richieste di interrupt è l'Interrupt Controller.

Anche nell'interrupt è presente un tempo di latenza per i dispositivi, che sarebbe l'intervallo di tempo che intercorre tra la richiesta di interruzione e il trasferimento del dato da/per I/O. Il tempo di latenza dev'essere il più piccolo possibile.

$$T_{Latenza} = T_{Istruzione} + T_{AttesaRoutine} + T_{InizioRoutineTrasferimentoDati}$$

Ogni dispositivo ha una priorità ad essere servito. La massima priorità viene assegnata al periferico con massimo  $TransferRate = \frac{1}{R_{lat}}$  e pertanto con minima latenza.

### Tipi di interrupt

Nell'80x86 ci sono 256 tipi di interrupt suddivisi in 4 gruppi:

**TRAP** sono interni es: generati dalla divisione per zero

---

**NON MASCH** non ignorabili es: accesso ad una locazione riservata

## **HW ESTERNI**

**SOFTWARE** es: INT n, usati per la richiesta di servizi

Gli ultimi 2 tipi sono MASCHERABILI (ignorabili) la CPU può non servirli se quello che sta eseguendo in quel momento è più importante.

Solo interrupt di un gruppo con priorità più elevata possono interrompere una routine di servizio dell'interrupt. Tutte le priorità sono modificabili tranne quelle del gruppo TRAP e NON MASCH.

Tramite il flag IF si possono abilitare e disabilitare gli interrupt (CLI porta IF=0, STI porta IF=1) L'IF viene automaticamente messo a zero (=disabilita interrupt) quando si attiva una routine di interrupt.

Gli INT NON MASCH sono attivati tramite il pin NMI sono usati per eventi tipo la caduta di alimentazione o errore lettura memoria.

Gli INT HW/SOFTWARE sono attivati tramite istruzioni software INT n, n tra 0-255.

TRAP rilevano situazioni anomale overflow e divisione per zero.

Esempio di Interrupt:

**0** → Divide per zero

**1** → Single Step

**3** → Breakpoint

**4** → Overflow

... Altri

Gli INT 1 e INT 3 vengono usati spesso nel debugging.

Protocollo Interrupt:

1. Un dispositivo invia una richiesta di interrupt su INTR
2. La CPU rileva la presenza del valore 1 su INTR
3. la CPU invia un impulso su INTA per segnalare che l'interrupt è stato rilevato
4. la CPU invia un secondo impulso su INTA per chiedere al dispositivo di scrivere sul DBUS il numero (n) del tipo di interrupt

- 
5. la cpu legge dal DBUS il valore  $(n) = 1$  byte
  6. la cpu salva nello stack i flag, registri e indirizzo di ritorno CS:IP
  7. la cpu azzerà IF e TF
  8. la cpu accede all'elemento  $4*n$  (=indirizzo su 32 bit) nell'IVT (interrupt vector table)
  9. viene attivata la corrispondente procedura di servizio

(figura pag 28)

PC INTERRUPTS (8259/APIC) L'8259 è stato progettato per minimizzare il software ed i tempi di risposta per la gestione dei livelli multipli di interrupt a diverse priorità

gestisce fino a 8 livelli di interrupt vettorizzati è incorporato nel SOUTH BRIDGE

Funzionamento:

1. Un segnale di richiesta di interrupt imposta il settaggio del bit  $IR(r)$  r da 0 a 7
2. l'8259 valuta le richieste e manda un segnale INT alla cpu
3. la cpu conferma la richiesta ed invia un primo segnale INTA
4. la richiesta a priorità più alta viene selezionata settando il bit  $IS(r)$  e resettando  $IR(r)$
5. la cpu invia un secondo impulso INTA
6. l'8259 invia sul DBUS il codice del dispositivo che ha fatto richiesta di interruzione
7. il ciclo di interrupt è concluso resettando  $IS(r)$  in modo automatico o con istruzione EOI (End of Interrupt)

L'8259 è programmato attraverso due tipi di parole di comando pilotate dalla cpu: ICW(Inizializza) inviate una volta sola OCW(Opera) inviate in qualunque fase del programma Entrambe sono composte da 9 bit settabili. Tralasciando la composizione della parole di comando in bit vediamo quelle principali:

**ICW 1** → resetta Interrupt Mask Register

---

**ICW 2** → la cpu determina i tipi di interrupt corrispondenti agli 8 segnali

**ICW 3** → specifica se il segnale è DISP. NORMALE, 8259 master, 8259 slave

**ICW 4** → specifica il buffer mode Automatico o EOI

**OCW 1** → permette di caricare il registro IMR settando i bit di esso a 1 (solo alcuni) e quindi si maschera il canale di interrupt IR(r) in poco serve ad abilitare e disabilitare gli interrupt

**OCW 2** → serve a gestire la rotazione di priorità

Di default l'8259 funziona in FULLY NESTED MODE.

Funzionamento generale: quando la cpu abilita una richiesta di interrupt, l'8259 calcola la richiesta a priorità più alta, il corrispondente indice del vettore delle interruzioni (n) e posto su DBUS e il bit IS(r) è settato.

Fintanto che il bit IS(r) è settato tutte le richieste di interrupt a priorità più bassa sono disabilitate.

→ solo le richieste a priorità più alta generano una richiesta di interruzione

→ in modo AEOI=1 l'8259 resetta IS(r) dopo il fronte di salita del secondo INTA

→ in modo AEOI=0 l'EOI viene mandato come ultima istruzione.

2 TIPI di EOI:

**specific EOI:** specifica IS(r) da resettare

**non specific EOI:** viene resettato il bit IS(r) di livello più alto

Può capitare che diversi dispositivi facciano una richiesta e abbiano la stessa priorità in questo caso conviene inviare l'OCW2 settato per priorità rotanti così appenda uno viene servito, gli viene assegnata la priorità più bassa questo evita che dispositivi con Tlat più alto non vengano mai gestiti.

Funzionamento FULLY NESTED MODE: usato nel caso di 8259 in cascata.

---

E' possibile espandere il sistema di controllori di interruzione fino a gestire 64 livelli di interruzione mediante l'uso di un master e 8 slave. Il master controlla gli slave mediante le tre linee CAS. Nel caso AEOI=0 bisogna mandare un segnale EOI per il master e uno per lo slave opportuno.

(figura pag 32)

1. Quando uno slave ha una richiesta di interruzione su un suo livello, invia una richiesta al pin IR del master mediante il segnale INT
2. tale richiesta è inoltrata alla cpu (deve essere a MAX priorità e NON MASCH)
3. quando la cpu invia il primo segnale, il MASTER setta IS(r) e pulisce IR(r) e legge il registro ICW<sub>3</sub> per sapere se tale richiesta proviene da uno slave oppure no.
4. Se la richiesta non proviene dall 8259 slave il master invia sul DBUS il contenuto dell' ICW<sub>2</sub>
5. se la richiesta proviene da uno 8259 slave il master piazza il numero del livello IR sulle linee CAS
6. il segnale INTA è ricevuto da tutto gli slave, ciascuno slave controlla il proprio ID con il numero letto dal CAS, se c'è corrispondenza riconosce che l'INTA è diretto a lui
7. lo slave selezionato setta IR(r) pulisce IRR e pone sul DBUS l'indirizzo della vector table contenuto nella propria ICW<sub>2</sub>
8. si manderà poi un duplice EOI

Le INTERRUZIONI MASCHERABILI sono gestite da 2 dispositivi 8259A(master-slave) Il personal computer è in grado di ricevere 15 tipi di interruzioni 7 sul master e 8 sullo slave. Gli 8259A sono inizializzati dal BIOS. DMA Serve a gestire in un altro modo le interruzioni. Taglia fuori la CPU (solo in parte).

Per la gestione in DMA è necessario che l'architettura preveda:

- cicli specifici di bus
- arbitraggio bus
- BIU ESTERNA al processore (DMA controller)

---

(figura pag 34)

Il DMA chiede il bus non di interrompere l'istruzione eseguita dalla CPU. Non viene interrotta l'istruzione, semplicemente il DMA si mette in mezzo, lavora nei cicli di bus e serve il disp. I/O al posto della CPU. Il trasferimento di un dato da MEM  $\leftrightarrow$  I/O avviene tramite due istruzioni:

```
MOV AH, mem (lettura memoria)
OUT io, AH (scrittura memoria)
```

nello stesso ciclo di bus deve essere selezionata sia la memoria sia l'interfaccia I/O.

- la memoria è selezionata mediante ABUS
- il periferico è selezionato mediante un circuito HW

Si passa così da 2 cicli a un solo ciclo(grazie al DMA)

Ogni canale di DMA contiene: REG. INDIR (16 bit), CONT. CICLI(n trasferimenti), REG di MODO

Quando un dispositivo DMA desidera acquisire il controllo del bus, porta ad HOLD=1, a questo punto la cpu terminato il corrente ciclo di bus mette ABUS e CBUS=Z e mette HLDA=1. Quando il dispositivo DMA rilascia il bus riporta HOLD=0.

L'8237 (DMA) è programmato con:

- indirizzo di partenza
- contatore N-1 (N numero di trasferimenti)
- inoltre va definito il tipo di trasferimento SINGLE o BLOCK MODE e il tipo di ciclo R/W

Se è single mode trasferisco un singolo byte per volta e ogni volta devo richiedere il possesso del bus.

Se è block mode (BUFFERIZATO) la richiesta viene fatta e tenuta aperta fino alla fine del buffer.

Al termine del trasferimento viene settato un bit(TC e inviato un segnale, al fine di capire quando il trasferimento complessivo sul canale è terminato.

---

## Gestione Modo Protetto - Memoria Virtuale - Paginazione

In MODO REALE gli indirizzi, indirizzano direttamente la memoria fisica seppur mediante un modello di segmentazione della memoria fisica stessa mediante 20 bit <sup>1</sup>

In *modo protetto* (multitasking) gli indirizzi sono espressi su 46 bit segmentazione della memoria logica(o virtuale).

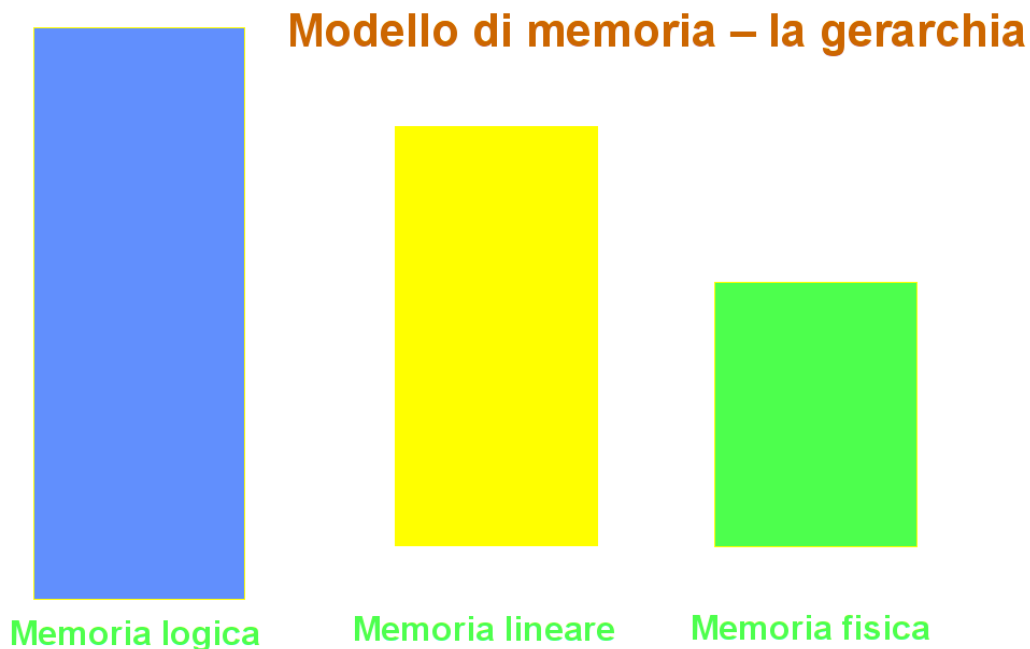
$$\text{Reg Seg}(14 \text{ bit}) + \text{Offset}(32/16 \text{ bit})$$

### Gestione indirizzamento logico (virtuale) della RAM

In modo protetto ogni segmento risulta essere pari a 4GB <sup>2</sup>

1. il registro di Segmento 14 bit specifica il segmento
2. l'offset determina l'indirizzo all'interno del segmento

**Figura 1:** Modello di memoria (gerarchia)



---

<sup>1</sup> Con 20 bit è possibile indirizzare 1MB  
 $2^{20}$

---

La memoria logica è formata da  $16K^3$  di elementi e ogni segmento è profondo 4GB. Non è detto che di ogni segmento si usano tutti i 4GB anche perchè la memoria Lineare è profonda 4GB quindi non ci sta più di un solo segmento totalmente pieno. Quindi di ogni ogni segmento si riempie solo una parte dei 4GB.

### Da LOGICA a FISICA :

1. Si mappa(SEGMENTAZIONE) la memoria Logica nella Lineare.<sup>4</sup>  
La memoria logica è divisa in 2:
  - GM (parte alta, risiedono i processi di sistema)
  - LM (risiedono i processi utente)
2. Paginazione da lineare a fisica. La memoria lineare viene istanziata in tante pagine che vengono messe nella fisica.
3. Nella memoria lineare si inseriscono delle porzioni della memoria Logica

### Memoria (definizioni):

#### LOGICA

spazio di memoria visibile dall'architettura del processore (assembler)

#### LINEARE

spazio di memoria gestito dall'architettura mediante un indirizzamento lineare.

#### FISICA

spazio di memoria direttamente indirizzabile dall'ABUS

#### REALE

memoria fisica realmente disponibile

Si è passati da 16 bit a 32 bit di indirizzamento in questi due modelli si usa la SEGMENTAZIONE, come da figura 2 . Da poco si è passati alla configurazione di 64 bit in questo caso la *segmentazione* non serve più: 14bit + 64 bit è improponibile<sup>5</sup>

---

<sup>3</sup> $2^{16}$

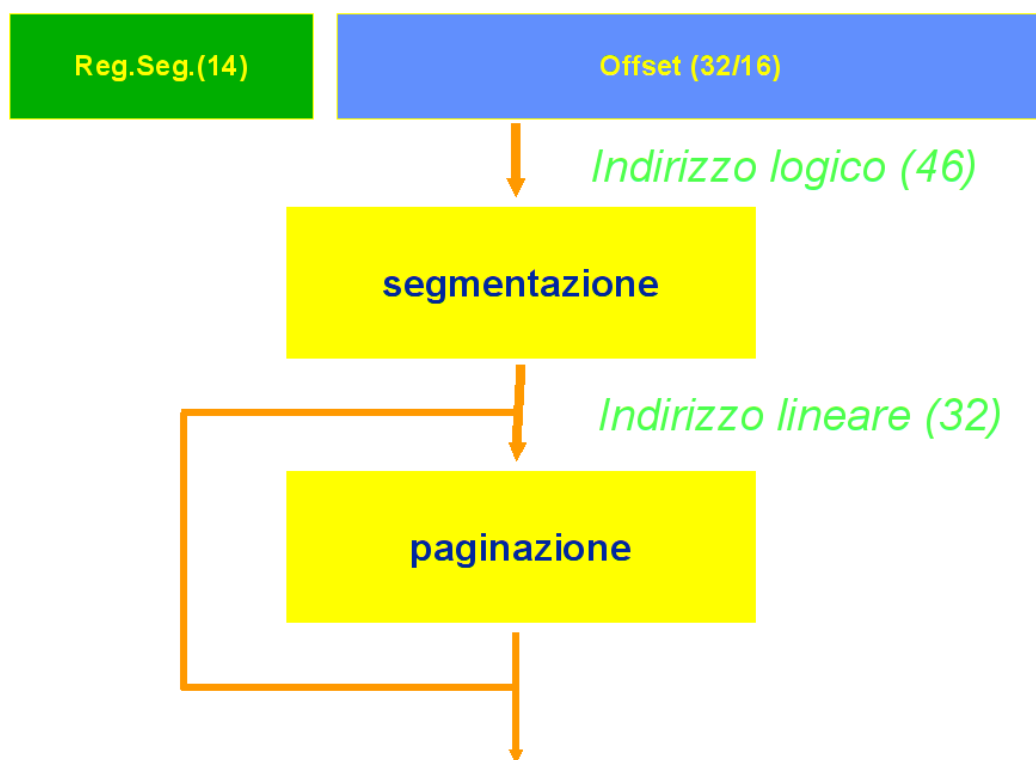
<sup>4</sup>La memoria lineare potrebbe anche essere la memoria fisica se di uguale dimensione

<sup>5</sup>La memoria lineare è molto profonda.



---

**Figura 2:** Determinazione indirizzo



---

La segmentazione e la paginazione eseguono una traduzione di indirizzi: ci sono quindi delle tabelle. Queste tabelle sono salvate in un sistema cache dedicato in modo tale che si va veloce a leggerle e non si rallenta la pipeline. *Segmentazione e Paginazione* lo si fa per tutte le istruzioni tranne per i movimenti tra registri.

Ogni segmento ha un descrittore. Questi descrittori sono contenuti in una tabella divisa in due. Il descrittore descrive il segmento e viene creato quando viene creato il segmento. Le due tabelle sono GDT e LDT quelle descritte prima.

#### **Contenuto del *descrittore*:**

Ogni descrittore è  
composto da 8 byte

- indirizzo di testa
- profondità (limite)
- attributi segmento
- livello di privilegio da 0 a 3

### **Mapping tra indirizzo logico e lineare**

Dal Reg. Seg ricavo il segmento interessato e il suo descrittore ne estraggo l'indirizzo di testa (di 32 bit) lo sommo all'offset (di 32 bit) —>

$$Reg.Seg + Offset = Ind.Logico$$

Da questa somma ottengo l'indirizzo lineare su 32 bit. Un volta ottenuto l'indirizzo lineare bisogna controllare la dimensione ossia confronto se la quantità di byte del Seg da mettere nella memoria Lineare non eccede il limite assegnato a quel dato processo contenuto in Seg.

Questo limite è scritto nel descrittore. Se il valore eccede l'HW blocca la mappatura.

#### **Attributi del descrittore:**

**PRESENZA** indica che il segmento è caricato nella mem. Lineare

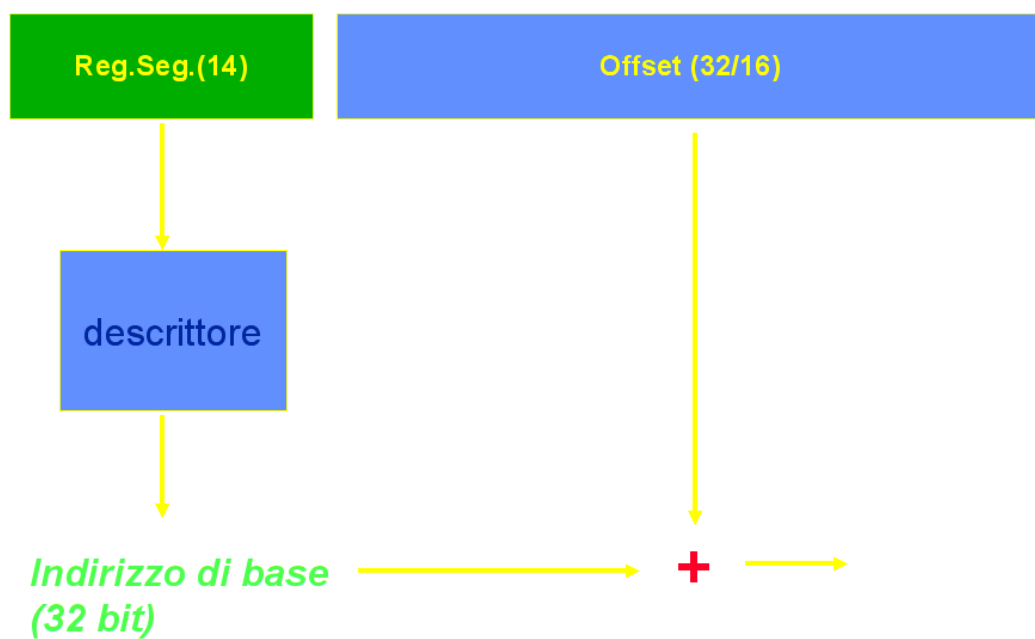
**DPL** privilegio

**SYSTEM** indica se il segmento è di dati, codice, R/W

---

**Figura 3:** Mapping tra indirizzo logico e lineare

## Mapping tra indirizzo logico e lineare



---

**EXECUTE** controlla il segmento è dati o codice in esecuzione

**R/W**

**ACCESSED** serve per lo swap quando la mem. Lineare è piena. Si va a vedere il valore di Accessed per vedere se quel dato segmento è stato usato oppure no, se non è mai stato usato lo si elimina e lo si sostituisce con un nuovo segmento.

Ad ogni Reg. Segmento vi è associata una piccola cache dedicata *8 byte* in cui è contenuto il descrittore di quel segmento.

Ad ogni entità (processo, codice, dati) è attribuito un livello di privilegio tra 0 e 3 (0 è MAX)

### Le leggi dei privilegi:

**R1:** Qualità dei dati: un processo può accedere a dati allo stesso livello o a livelli più alti numericamente.

**R2:** Affidabilità del codice: un segmento di codice può accedere ad un altro segmento solo se allo stesso livello o a livello più basso numericamente di quel segmento che vuole leggere.

MOV mem, AX → MOV dpl=3  
→ mem dpl=1            L istruzione non si può eseguire.

## Paginazione

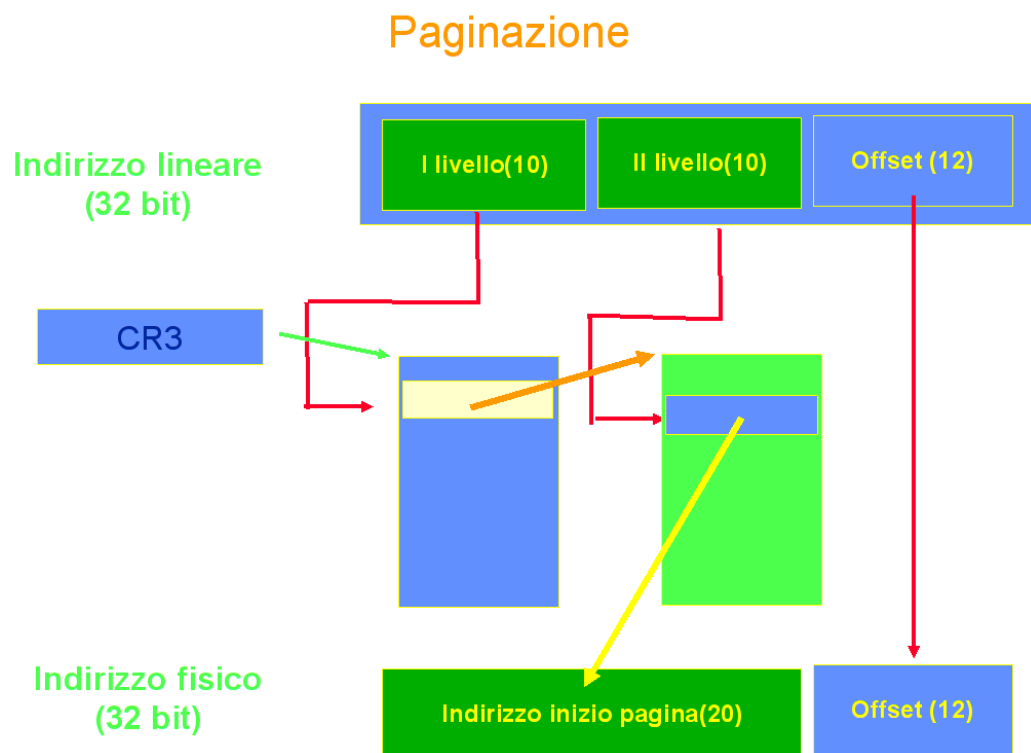
Ogni pagina è di  $4K^6$  fino a 4 MB.

1. l'indirizzo di testa della pagina è calcolato dal sistema di paginazione a partire dai 20 bit più significativi dell'indirizzo lineare e produce i 20 bit più significativi dell'indirizzo fisico.ù
2. l'offset nella pagina è prelevato direttamente dall'indirizzo lineare (12 bit)

---

<sup>6</sup>2<sup>12</sup>

Figura 4: Diagramma paginazione



---

### **Paginazione:**

1. si legge il registro CR<sub>3</sub> (reg paginazione) questo registro punta alla testa di una tabella di 32 bit profonda  $2^{10}$ .
2. si leggono i primi 10 bit dell'ind lineare così si trova una della 1024 entry della prima tabella qui dentro è presente un indirizzo.
3. Si legge l'indirizzo di quella entry che punta ad un'ulteriore tabella profonda 1024 entry, si va a leggere gli altri 10 bit dell'ind lineare e si trova la seconda entry nella seconda tabella.
4. Il contenuto della seconda entry mi dà i 20 bit dell'ind inizio pagina nella mem fisica. Gli altri 12 sono ottenuti direttamente dall'offset della mem lineare.

### **Struttura:**

- Ogni tabella è costituita da 1024 elementi.
- Ogni elemento è costituito da 8 byte
- Ogni elemento contiene 20 bit di indirizzo e un certo numero di bit attributo

### **Considerazioni:**

- Le porzioni delle tabelle sono contenute in una cache dedicata nella cpu.
- Architettura a 3 livelli:
  - mem logicamente
  - mem lineare
  - mem fisica

Può essere ridondante, l'INTEL usa un'architettura a 3 livelli ma senza la segmentazione. I privilegi vengono incorporati nella paginazione, negli attributi delle entry di 2 livello:

0,1,2 = 0 Super

3 = 1 User

Comunque è il sistema operativo che può scegliere se usare solo la paginazione o anche la segmentazione.

---

## Modo Protetto

### Esempio:

MOV AX, [EBX]

DS=200 EBX=155  
Ax è di 2 byte ind  
logico(virtuale) =  
DS:EBX

Voglio accedere al 200<sup>esimo</sup> segmento.

Essendo ogni segmento di profondità 4GB, all'interno del 200<sup>esimo</sup> segmento voglio i primi 2 bytes all'indirizzo 155 e 156.

Quando carico in DS il valore 200 viene automaticamente creato il descrittore e caricato nella cache specifica.

All'interno del descrittore ci sarà anche l'indirizzo di testa della memoria lineare per quello specifico segmento

es: BA = 10000

Quindi farò come già detto indirizzo di testa più offset = 10155.

Questo è l'indirizzo di partenza del dato nella LINEARE.

Bisogna poi confrontarlo con il LIMITE, se il limite è es: 300 allora 155 < 300 quindi: OK carico segmento nella lineare altrimenti genero una TRAP.

Poi rimane da fare la paginazione: L'indirizzo della LINEARE è su 32 bit = 10155 → mem fisica usando le tabelle di paginazione (2 tabelle) per i primi 20 bit mentre i 12 bit di offset vengono copiati direttamente.

Dopodichè vado a vedere i privilegi, è ovvio che per eseguire questa istruzione l'istruzione stessa deve avere un DPL numerico minore di quello della locazione di memoria [EBX] altrimenti non può essere eseguita.

**Considerazioni** Nei S.O. Odierni la segmentazione è disabilitata nel senso che si definiscono segmenti di 4GB sovrapposti e tutti puntano all'indirizzo top della memoria lineare: SEGMENTAZIONE bypassata.

### Passaggio al modo protetto

1. inizializzare GDT con 3 descrittori CODE, STACK e DATA
2. inizializzare IDT (Interrupt Description Table)
3. switch al modo protetto settando il bit PE(Protected Enabled)
4. azzerare coda istruzioni con JMP FAR

---

5. caricare DS, SS i selettori

L'inizializzazione della IDT viene fatta delimitando le istruzioni di codice con:

CLI

.....  
.....  
.....

STI

In modo protetto è essenziale attivare la paginazione però prima devo attivare la tabella di paginazione (cioè caricarle nella specifica cache) e scrivere nel registro es: CR<sub>3</sub> l'indirizzo di testa delle tabelle di paginazione.

La MEM. LOGICA/ LINEARE che contiene i descrittori è divisa in due tabelle GDT e LDT.

Nella GDT che sta verso gli indirizzi più grandi FFFFh c'è il bios, kernel processi sistema e occupa metà della memoria logica l'altra metà è occupata dalla LDT dove ci stanno i processi utente.

- Soluzione 1 (UNA SOLA LDT) caso normale. Nella lineare occupa 2GB
- Soluzione 2 (PIU' LDT) una per ogni processo di ampiezza 2GB

### **TSSR (Task State Segment register)**

1. Processo attivo
2. Arriva interrupt
3. Si salva lo stato del processo in una tabella assieme ai registri usati dalla CPU, stack e flag, le pagine, indirizzi I/O che sono visti in modo diverso da processo a processo, registro CR<sub>3</sub> e tabelle di paginazione.

Tutto questo viene salvato totalmente via hardware in uno spazio riservato specificato appunto all'interno dei registri TSSR.

## **Evoluzione architettura Pentium 80x86 /64**

**Multiprocessor chip** : ci si è spostati sul parallelismo dei threads

**architetture SMP on chip** : sincronizzazione cache e cpu sul chip



---

**memoria fisica** : ABUS esteso a 52 bit contro i 32 bit di prima

**registri estesi** : registri offset a 64 bit contro i 32 bit di prima

**gerarchia memoria:** solo paginazione, NO SEGMENTAZIONE Gestisce tutto la paginazione con 4 livelli di tabelle di paginazione.

## PAE (Physical Address Extension)

### Estensione dell'ABUS di 4 bit

A partire dal pentium 2 l' *ABUS* è stato portato a 36 bit sempre con parallelismo del *DBUS* a 64 bit.

Poiché gli indirizzi lineari sono ancora su 32 bit, la memoria fisica è gestita mediante una diversa organizzazione delle tabelle di paginazione, che hanno entry da 64 bit al posto di 32 bit.

Le 2 tabelle di paginazione hanno ogni entry a 64 bit: tabelle così larghe si ottengono usando invece di  $10 + 10 + 12$  un'organizzazione  $2 + 9 + 9 + 12$  (pagine da 4K).

I primi 2 bit creano una tabella con 4 entry puntate da CR3 i primi 9 bit servono per trovare l'indirizzo nella 1 tabella mentre i successivi 9 bit servono per trovare l'indirizzo nella seconda tabella puntata dalla prima da lì poi si prendono 24 bit + 12 che formano la dimensione dell'ABUS.

Quindi prima la mem fisica era formata da 20 + 12 ora invece si è arrivati a 24 + 12 si possono gestire più indirizzi.

Se si ha una memoria di dimensioni elevate si possono aumentare le dimensioni delle pagine portandole a 2MB—> Mem lineare organizzata 2 + 9 + (12+9) di offset così facendo si è ridotta la parte di indirizzo delle pagine ma si è aumentata la dimensione delle pagine.

posso trasportare più dati rispetto a prima.

**ESTENSIONE A 64 bit** Si usano indirizzi denominati RAX.

Esempio:

MOV RAX, mem

- prevede un indirizzo lineare su 48 bit
- prevede un indirizzo fisico su 40 bit

Il passaggio da 32 a 48 nel LINEARE e da 32 bit a 40 bit nella Fisica è ottenuto come descritto prima da una riorganizzazione delle tabelle di paginazione. La memoria lineare è su 64 bit ma se ne usano solo 48 gli altri 12 bit non sono altro che la propagazione dei 48 bit.

Compatibilità con il futuro

---

**SMP Dual Processor** Man mano che si va avanti si metteranno sempre più livelli di cache interni alla CPU e uno fuori in comune. Cioè se si hanno solo 2 livelli di cache la soluzione migliore è usare una cache L1 per ogni CPU e una L2 comune perchè così facendo la L2 può assumere dimensioni notevoli.

Stessa cosa se si usano 3 livelli di cache: L1, L2 divise L3 comune.

Non solo la dimensione ma anche per quanto riguarda i processi che usano dati comuni, questi dati comuni invece di essere parcellizzati in entrambe le cache vengono messi in una cache comune L3 (in questo caso) e manipolati dai due processi.

L'uso del livello più alto come cache comune porta anche ad eseguire meno aggiornamenti sui dati in cache.