

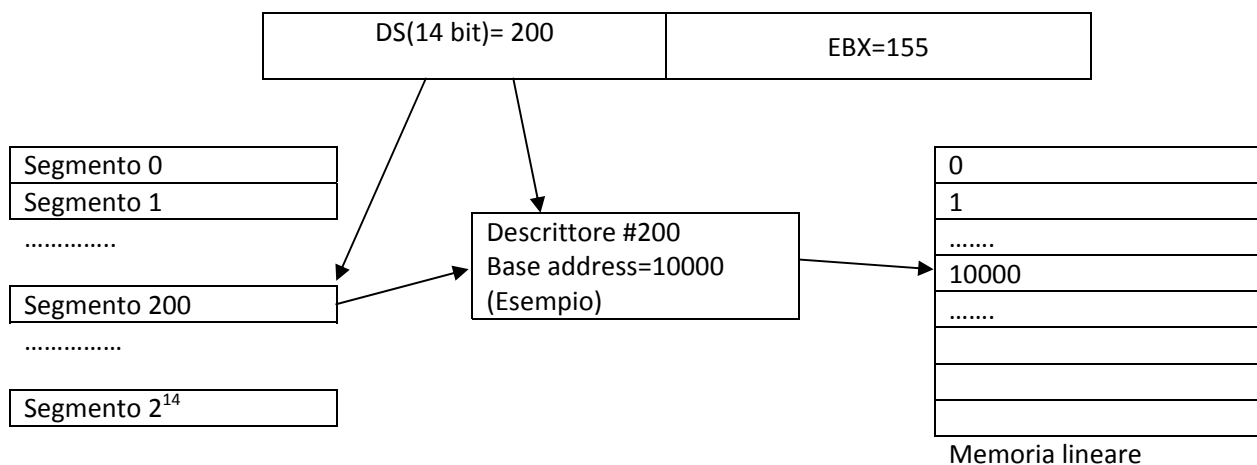
Esempio esplicativo sul meccanismo segmentazione/paginazione

Ipotesi:

- Modo protetto
- DS= 200
- EBX= 155

L'istruzione in esame è **MOV AX,[EBX]**

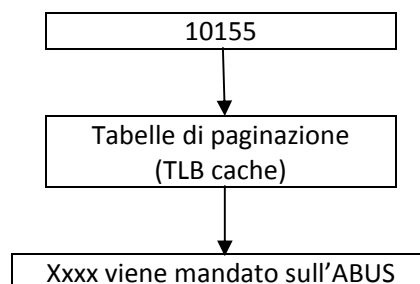
- L'indirizzo logico(virtuale) oggetto dell'istruzione è **DS:EBX = 200:155**



Memoria logica

- L'indirizzo lineare è: $10000(\text{Base Address}) + 155(\text{EBX}) = 10155$

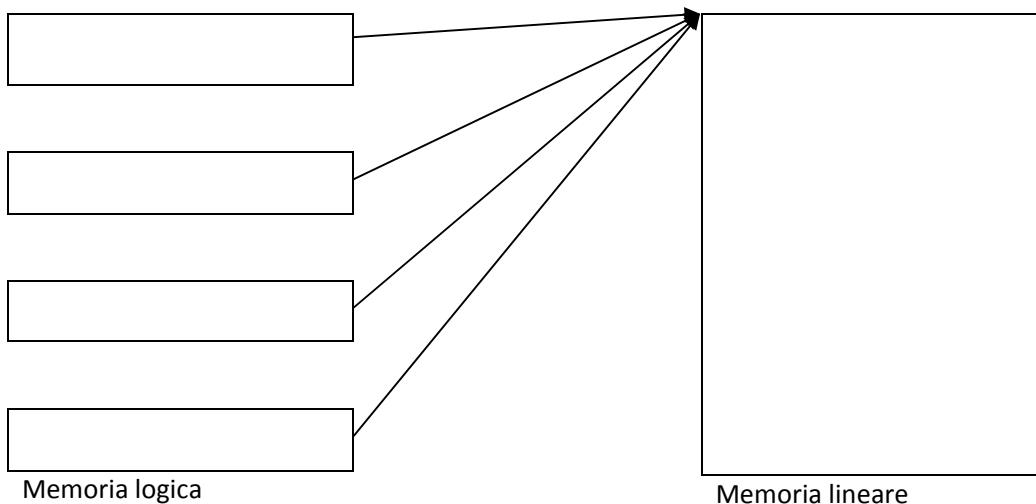
Su quest'indirizzo vengono poi effettuati dei controlli di congruità con la lunghezza dei segmenti.



Il valore Xxxx dipende dal contenuto delle tabelle di paginazione

Disabilitazione della segmentazione

Nell'Intel 80x86 la segmentazione non è disabilitabile direttamente in maniera esplicita, però è possibile aggirare questo problema facendo sì che tutti i segmenti puntino direttamente (attraverso le tabelle dei descrittori) alla stessa area di memoria lineare definendo segmenti molto ampi (4GB) e sovrapposti.



In questo modo il mapping e la gestione dei privilegi sono demandati interamente al meccanismo della paginazione. I registri di segmento vengono così inizializzati una sola volta al boot del sistema operativo.

Una memoria così gestita è chiamata **memoria lineare paginata**.

Questa tecnica è diffusa nei sistemi operativi (Es. Windows, Unix like).

Contenuto della tabella di paginazione

Proprio perché la memoria può essere gestita in questo modo, c'è bisogno che la paginazione, oltre a provvedere al mapping sulla memoria fisica, preveda dei meccanismi per gestire alcuni compiti della segmentazione (privilegi, accessibilità, accesso).

Per coprire queste necessità, in ogni linea della tabella di paginazione (di II livello) compaiono 4 campi:

- Indirizzo inizio pagina: 20 bit che indicano i bit più significativi dell'indirizzo di inizio pagina (che verrà poi sommato all'offset per generare l'indirizzo finale)
- S/U: [System/User] 1 bit che indica il tipo di privilegio. I privilegi possibili a livello di paginazione sono 2: System, equivalente ai livelli 0,1,2 della segmentazione e User, equivalente al livello 3 della segmentazione
- R/W: [Read/Write] 1 bit che indica la modalità in cui è accessibile quella pagina (Lettura/Scrittura)
- A: [accessed] 1 bit che segnala se è stato fatto accesso alla pagina per modificarla

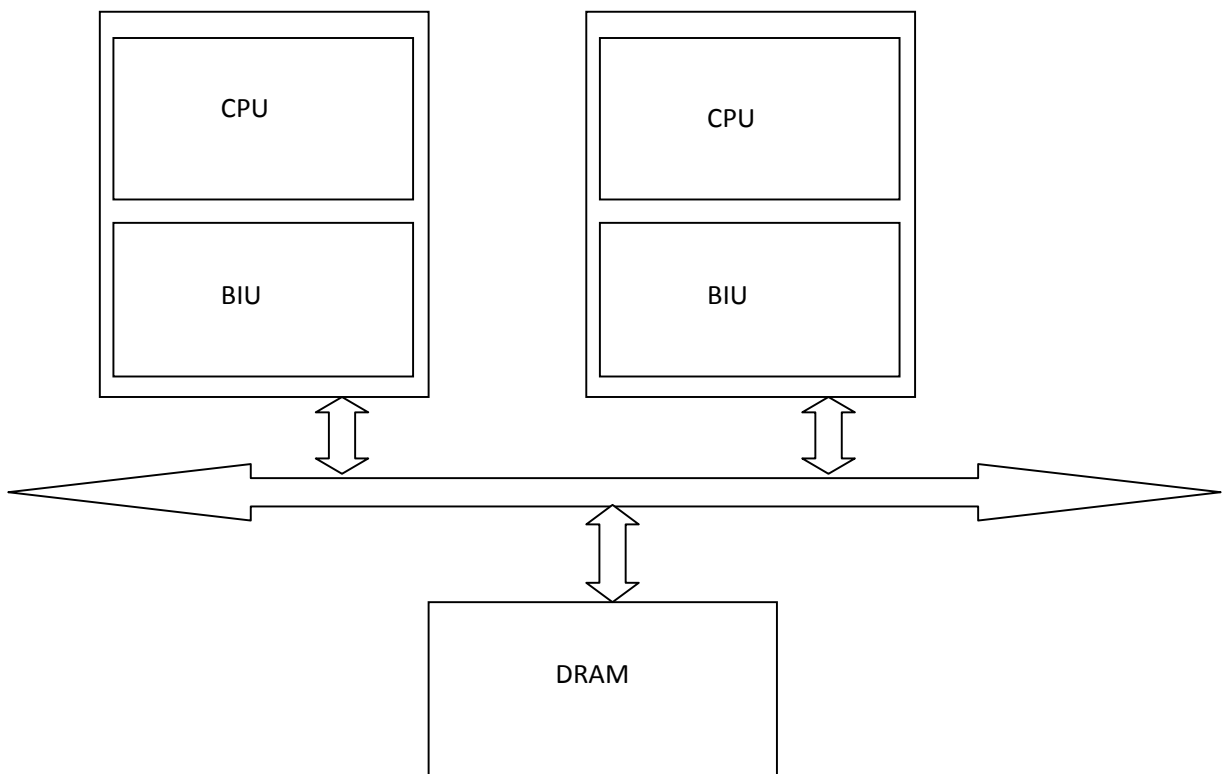
Nelle entry della tabella di primo livello, invece, compaiono sono i 20 bit dell'indirizzo di testa della tabella di secondo livello corrispondente.

Evoluzioni dell'architettura

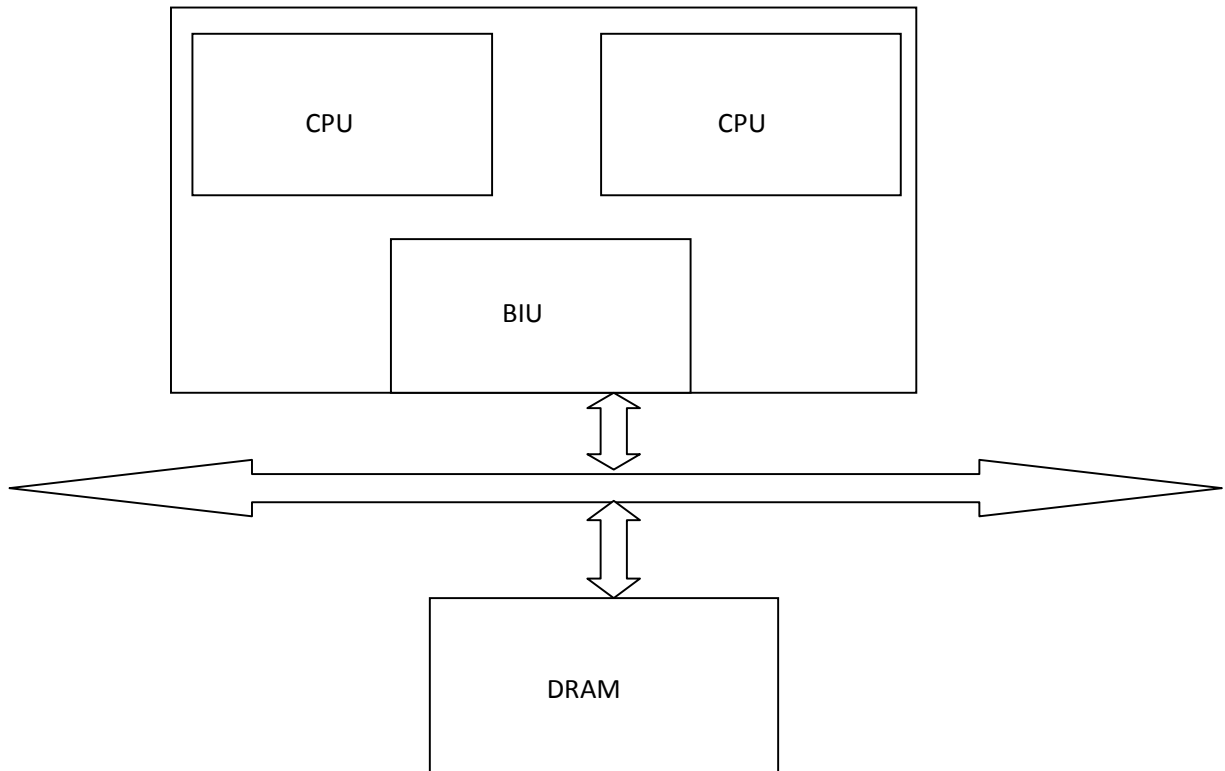
- **Multiprocessor chip:** si tende ad aumentare il numero di cpu e di conseguenza si tende a prediligere il parallelismo dei thread
- **Architettura SMP on-chip:** la sincronizzazione dei processori e della cache viene gestita direttamente sul chip stesso
- **Memoria fisica:** il bus indirizzi viene esteso a 52 bit, aumentando lo spazio di memoria direttamente indirizzabile verso l'ordine dei TB
- **Registri estesi:** il parallelismo dei registri di offset viene portato a 64 bit (RAX), permettendo di fatto di percorrere tutta la memoria con il solo offset
- **Gerarchia della memoria:** viene abolita la segmentazione e si utilizzano 4 livelli di tabelle di paginazione anziché due. Tutte è gestito dalla paginazione (privilegi, mapping...)
- **FSB:** nel Pentium 4 il bus è a 100Mhz double o quad pumped, anche se data la cache on-chip la velocità del bus assume minor importanza

SMP vs Hyper Threading

SMP:



Hyper threading:



Le architetture SMP (*Symmetric multi processing*) sono caratterizzate dalla presenza di più processori fisici che hanno accesso tutti a tutta la memoria Ram di sistema; ogni processore esegue un thread o un processo diverso aumentando notevolmente il parallelismo d'esecuzione e parimenti le prestazioni complessive del sistema. L'introduzione dell'ambiente multiprocessore introduce numerose problematiche di gestione del parallelismo, della schedulazione dei processi e dei thread associati, della gestione di coerenza tra i dati delle cache e di servizio degli interrupt.

Al sistema operativo è quindi chiesto di schedulare correttamente i thread e i processi al fine di ottenere il migliore sfruttamento possibile di questa architettura; meccanismi hardware lo aiutano in questo compito. Stesso problema compare per gli interrupt, dove bisognerà scegliere quale processore deve interrompere la propria attività, quindi fermare l'esecuzione di un processo, e servire l'interrupt; la scelta ricade sul processore che sta eseguendo il processo a più bassa priorità, selezionato grazie a architetture hardware come gli APIC. Alla gestione della coerenza tra i dati delle cache è dedicata la sezione successiva.

Nelle architetture basate sulla tecnologia dell'Hyper Threading su di un singolo die convivono più unità esecutive che permettono l'esecuzione di più thread dello stesso processo simultaneamente. Quindi non è possibile eseguire più processi nello stesso momento, da che ne deriva una schedulazione orientata ai thread. Infatti a differenza delle architetture SMP ad essere duplicati non sono gli interi processori, ma semplicemente l'area dedicata all'esecuzione delle istruzioni, è compito poi del BIOS far credere al sistema operativo di trovarsi di fronte a due processori veri e propri.

Nelle intenzioni di Intel questa architettura fu lanciata con l'intento di far sì che a fronte di un aumento di solo il 5% dell'estensione del core (quindi con un aumento di consumo limitato e un aumento di prezzo limitato), si ottenesse un miglioramento del 30% delle prestazioni, ma l'arrivo dei processori dual e quad core e il mancato raggiungimento della soglia del 30% (i miglioramenti furono dell'ordine dell'appena il 10% in alcune applicazioni avanzate) decretarono il progressivo insuccesso della soluzione Hyper Threading. Ultimamente, con l'avanzata del mercato dei computer ultraportatili (e ultra economici) e con le problematiche di consumo ad essi collegate, l'architettura sta vivendo un periodo di riutilizzo (seppur con alcune evoluzioni, come il Simultaneous Multi Threading) nel mercato dei processori.

Gestione di coerenza della cache nei sistemi SMP

Nell'architettura SMP i processori condividono la stessa BIU e ognuno di loro ha una propria cache (L1 e L2).

La presenza di cache separate introduce il problema della coerenza dei dati ivi contenuti, infatti è possibile che nelle cache siano contenute linee le quali si riferiscono allo stesso dato in memoria. Da qui nasce la necessità di prevedere dei protocolli per far sì che una modifica da parte di un processore su un dato comune alle cache sia acquisita anche dalle altre.

È risaputo che i due modi principali di gestione della cache sono il write-through e il write-back; l'uso di una o l'altra modalità influisce pesantemente nelle architetture a multiprocessore, dove ogni processore ha la sua cache e dove nelle varie cache si può fare riferimento più volte allo stesso dato, infatti l'uso del write-back (quindi scritture sul dato solo in cache, aggiornamento del dato in memoria solo in caso di FLUSHING, miss e istruzione WBINVD) non permette di tenere sincronizzate tra loro le linee di cache referenti allo stesso dato e appartenenti a processori diversi, invece l'uso della modalità write-through (quindi scritture sul dato sia in cache sia in memoria centrale) permette a tutti i processori di tenere allineate le proprie linee di cache, in quanto rende nota a tutti i processori (la memoria centrale è visibile a tutti) l'eventuale alterazione del dato.

La soluzione utilizzata è la combinazione di due protocolli: il protocollo di Snooping e il protocollo MESI.

Protocollo MESI

Questo protocollo è usato per la gestione dello stato della cache. Ad ogni linea di cache è associato uno stato definito su 2 bit (definendo quindi 4 stati possibili)

Gli stati possibili sono:

- M: Modified: il suo valore è diverso da quello in memoria, quindi il dato è aggiornato solo in cache (dirty)
- E: Exclusive: il dato è aggiornato con quello in memoria, ma è usato solo nella cache di quel processore
- S: Shared: il dato è aggiornato con quello in memoria, ma oltre ad essere contenuto in quella cache è contenuto anche nella cache di almeno un altro processore
- I: Invalid: il dato non è valido (Ad esempio il dato è aggiornato in memoria ma non in cache)

Protocollo di Snooping

Questo protocollo è usato per mantenere la coerenza dei dati condivisi nelle varie cache.

Ogni processore ha la possibilità di “annusare” (ENG: to snoop) l’indirizzo che transita in quel momento sull’ABUS per controllare se qualche processore sta facendo accesso in scrittura ad uno dei dati che la sua cache “privata” contiene. Se ciò avvenisse, il processore dovrebbe aggiornare il dato contenuto in cache con quello nuovo in memoria.

Il processore che sta per trasmettere sul ABUS un indirizzo invia un segnale di SNOOP per avvertire gli altri di controllare se sta per modificare un dato presente nelle cache di un altro processore. Il segnale di SNOOP può essere condizionato allo stato MESI della linea di cache (nel caso di linea Exclusive non ha senso far controllare agli altri processori).

Additional info comes from: wikipedia.it