

Significato dei bit di attributo nel descrittore di segmento

- 1) **Presenza:** indica se il segmento è caricato nella memoria lineare
- 2) **DPL:** indica il livello di privilegio
- 3) **System:** indica se il segmento è di dati/codice o di sistema (es. contiene delle tabelle di sistema)
- 4) **Execute:** nel caso si tratti di un segmento di dati/codice, specifica se si tratti di dati o codice
- 5) **Read/Write:** indica se il segmento può essere letto o scritto
- 6) **Accessed:** indica se il segmento è stato letto o scritto.

I valori di questi bit sono tenuti costantemente aggiornati dall'hardware stesso, al momento della fase di execution di una istruzione. Ciò fa intuire la necessità di poter accedere ai descrittori dei segmenti in maniera estremamente veloce. Siccome l'accesso al descrittore avviene per ogni esecuzione di una istruzione, il tempo di accesso deve essere inferiore al tempo di uno stadio di pipeline, ossia circa 1ns.

Questo vincolo temporale comporta l'impossibilità di collocare i descrittori sia nella memoria principale, sia addirittura nelle cache del processore. L'unica velocità adatta allo scopo si raggiunge associando una mini-cache ai registri dei vari segmenti, che si trovano all'interno della CPU.

Funzionalità del sistema di privilegi

A differenza di quanto accade nel modo reale, nel modo protetto ad ogni segmento è associata una certa gestione dei privilegi. Tale sistema di protezione è indispensabile per garantire la coerenza dell'ambiente stesso e per poter realizzare la programmazione concorrente. Nello specifico, il sistema di privilegi garantisce le seguenti due condizioni:

- 1) **Qualità dei dati:** un processo può accedere a dati con livello di privilegio pari o inferiore al proprio, secondo le seguenti relazioni:

$PL_{processo} \geq PL_{dati}$ (logicamente)

$PL_{processo} \leq PL_{dati}$ (numericamente)

L'inversione del segno nelle due disuguaglianze è semplicemente dovuta al fatto che il livello a privilegio più alto corrisponde al valore 0.

- 2) **Affidabilità del codice:** un segmento di codice può accedere ad un altro segmento solo se quest'ultimo ha un livello di privilegi pari o inferiore al suo.

Struttura di un file eseguibile

<i>Tabella dei segmenti</i>
<i>Tabella dei riferimenti</i>
<i>Entry point</i>
<i>C1</i>
<i>D1</i>
<i>D2</i>
<i>....</i>

Tabella dei segmenti: contiene le indicazioni sul numero dei segmenti, sulle loro lunghezze e su dove essi sono posizionati nel file eseguibile.

Tabella dei riferimenti: contiene le indicazioni su quali indirizzi calcolare, andandoli a sostituire poi nelle istruzioni interessate. Al momento della compilazione non è infatti possibile sapere l'esatta collocazione dei segmenti nella memoria, poiché essi sono allocati effettivamente solo al momento dell'esecuzione del processo. Il loader del sistema operativo deve quindi occuparsi di completare il codice con gli indirizzi effettivi, sostituendoli nei riferimenti non ancora risolti.

Si pensi ad esempio al caso di una chiamata Assembly ad una procedura di tipo "far". Non è possibile al momento della compilazione sapere l'indirizzo effettivo del segmento che contiene la procedura invocata.

Entry point: indica il punto di partenza del programma, specificato in Assembly tramite la direttiva .STARTUP

La paginazione

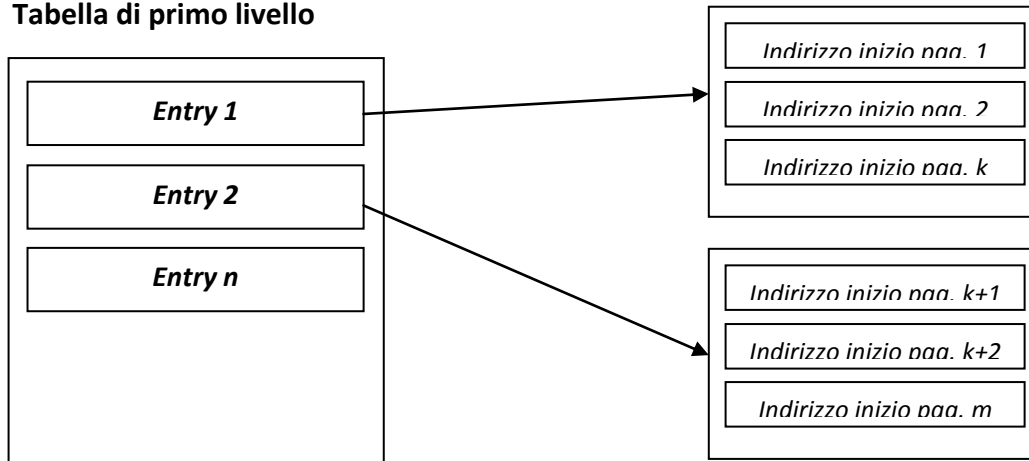
La paginazione è un meccanismo che permette di mappare la memoria lineare (generalmente più estesa) su quella fisica (generalmente meno estesa). Si basa sulla suddivisione della memoria lineare in porzioni dette "pagine". L'adozione di questo meccanismo pone subito due questioni basilari, ossia stabilire la dimensione delle pagine e come tradurre (ossia mappare) gli indirizzi.

Generalmente si utilizzano pagine da 4 KB, in modo da farne stare un buon numero nella memoria fisica. E' così possibile sfruttare il principio di località tra pagine che corrispondono a frammenti di memoria adiacenti.

Per il sistema di mapping tra indirizzi in memoria lineare e indirizzi in memoria fisica, in genere si utilizza una gerarchia a più livelli di tabelle, che contengono le traduzioni degli indirizzi. Nel caso pratico dell'architettura Pentium standard a 32 bit si utilizzano due livelli di tabelle.

Utilizzando questo approccio gerarchico si hanno notevoli vantaggi sia dal punto di vista della quantità di memoria occupata, sia da quello dei tempi di traduzione. Potrebbe infatti verificarsi il caso in cui, utilizzando una tabella unica molto estesa, venga sprecato molto spazio per allocare entry non utilizzate.

Tabella di primo livello



Di conseguenza anche il mapping degli indirizzi avviene a livelli: l'indirizzo lineare viene suddiviso in una parte iniziale lunga 10 bit (primo livello), che indica la entry nella tabella di primo livello, in una seconda parte lunga 10 bit che specifica la entry della tabella di secondo livello puntata dall'indirizzo di primo livello, e infine in una terza parte lunga 12 bit, che contiene l'offset.

Al termine del processo di mapping si avrà quindi in uscita un indirizzo fisico su 32 bit, di cui i primi 20 bit corrispondono all'indirizzo di inizio pagina, e i rimanenti 12 bit indicano l'offset all'interno della pagina.

Nel caso in cui la segmentazione non sia abilitata, le gestioni dei privilegi e dei permessi vengono delegate alla paginazione. Siccome le entry delle tabelle di secondo livello sono più estese del necessario, nello spazio rimanente vengono memorizzati i bit di privilegio della relativa porzione di memoria.

La cache TLB

Anche per il meccanismo della paginazione si presenta il problema dei tempi di accesso alle tabelle di mapping, questione già emersa nel caso dei descrittori dei segmenti. Analogamente infatti l'accesso alle tabelle avviene in ogni fase di decodifica di un'istruzione, e pertanto il tempo impiegato da tale operazione deve essere compatibile con la durata di uno stadio di pipeline. Ciò preclude sia l'utilizzo della RAM che quello delle cache per memorizzare le tabelle, e costringe all'utilizzo di particolari registri interni alla CPU, che costituiscono la cosiddetta "cache TLB".