di Pierluigi Panunzi

I task e l'ambiente multi-tasking

seconda parte

Dopo aver parlato diffusamente dei task e della loro «carta d'identità» costituita dal TSS («Task Status Segment») e dopo aver visto come quest'ultimo rappresenti in pratica la zona di memoria in cui viene salvato lo stato del task correntemente in esecuzione, per effetto di un «task switch», in questa puntata proseguiremo nell'analisi della logica che sta alla base del task switch.
Ma prima di procedere oltre, andiamo a conoscere un ennesimo rappresentante della famiglia dei descrittori, il «task gate», che è strettamente imparentato con gli altri «gate» che sin qui abbiamo conosciuto

Il «task gate»

Si tratta ancora una volta di una struttura formata da 4 word (vedasi la figura 1), che serve in tutti quei casi in cui si vuole effettuare un task switch (ne parleremo meglio nel seguito) in modo se vogliamo «indiretto», così come già succedeva per i «call gate», oppure (e questo è senz'altro più importante) quando il nostro task può essere attivato da più eventi.

In particolare un task può essere attivato allorché l'istruzione di JMP, CALL, IRET o un interrupt fanno riferimento ad un «task gate» piuttosto che direttamente ad un TSS: in questo senso l'attivazione di un task attraverso un task gate ha un processo meno diretto.

Infatti dalla figura 1 notiamo subito la presenza di un campo chiamato «TSS SELECTOR» che per l'appunto è proprio il selettore del TSS relativo al task in esame.

A parte poi i campi «UNUSED» (in quanto del tutto sovrabbondanti) e gli immancabili «INTEL RESERVED» destinati ad essere utilizzati dal 386, troviamo un «ACCESS RIGHTS BYTE» (già scisso nei suoi sotto-campi) in cui compare il pattern «00101» caratteristico di un task gate, oltre agli oramai ben noti bit «P» di «Present» e «DPL» («Descriptor Privilege Level»), rispettivamente legati alla presenza o meno in memoria del descrittore ed al livello di privilegio minimo per poter accedere al task gate.

A differenza dei TSS, che devono essere posti obbligatoriamente nella GDT («Global Descriptor Table»), i task gate possono essere posti a seconda delle necessità sia nella GDT sia nella LDT («Local Descriptor Table»): quest'ultimo caso è previsto proprio per le situazioni in cui è necessario che un certo task abbia un accesso, diciamo così, «privato» ad alcuni altri task, ma anche in questa situazione i TSS dei task «locali» devono sempre essere posti nella GDT.

Per quanto riguarda l'accesso ad un task (e cioè al suo TSS) per mezzo di un task gate, bisogna che il livello di privilegio del selettore di destinazione (l'EPL, «Effective Privilege Level», del TSS) sia numericamente minore o uguale del livello di privilegio contenuto nel campo DPL del gate stesso: sappiamo bene che un'eventuale violazione a questa regola comporta una segnalazione di errore da parte del sistema operativo.

Parlato dunque del task gate, ritorniamo indietro ad analizzare in quale modo può essere iniziato un task switch ed in quale modo viene portato avanti.

Il task switch

Per effettuare un task switch abbiamo quattro possibilità:
— tramite l'esecuzione di una istruzione CALL FAR o di una istruzione JMP FAR, cioè quelle istruzioni che contengono l'offset ed il selector della routine di destinazione: il task switch si avrà nel caso in cui il selector della locazione di destinazione non fa riferimento ad un segmento di codice ma viceversa si riferisce («punti») ad un TSS descriptor. In tal caso l'offset contenuto nell'istruzione viene ignorato in quanto l'entry point del task sarà presente come CS:IP nel TSS.
— Quando viene eseguita un'istruzione IRET allorché nella parola di flag il bit NT

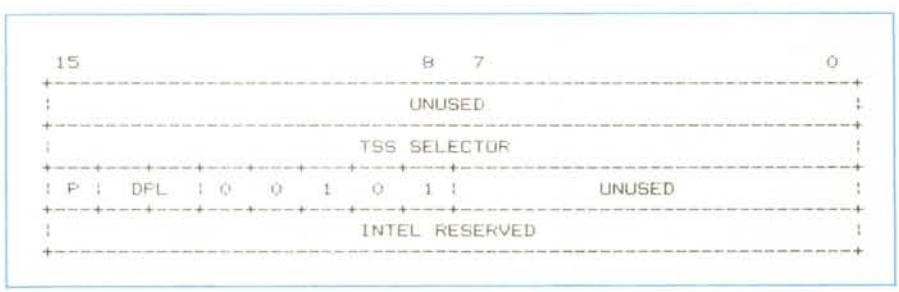

Figura 1 - Struttura di un «task gate», in cui si riconoscono i soliti campi, in particolare l'«Access Rights Byte» in cui compare il valore binario 00101 caratteristico di un task gate. In questo caso ben due campi risultano inutilizzati perché ridondanti.

(«Nested Task») è pari ad 1: questo significa che sta terminando un task il quale a sua volta ne aveva interrotto un altro, per effetto di un interrupt. In questo caso l'indirizzo a cui fare ritorno non è immediatamente disponibile: in particolare il selector del TSS del task da riattivare è posto nel TSS del task in corso di esecuzione, proprio nel campo «back link» e una volta risalito al TSS del task da riattivare si trova subito l'indirizzo della prossima istruzione da eseguire, al solito in CS:IP.

— Tramite l'esecuzione di una istruzione di CALL FAR o di una istruzione JMP FAR dove questa volta il selector contenuto nell'istruzione stessa fa riferimento ad un «task gate», piuttosto che ad un descrittore di TSS: in questo caso all'interno del task gate troveremo il selector del TSS descriptor relativo al task da attivare.

Anche in questo caso l'offset contenuto nell'istruzione CALL o JMP viene ignorato, in quanto ancora una volta l'entry point del task è ricavabile dal TSS.

— Quando viene generato un interrupt il cui interrupt vector, all'interno della IDT («Interrupt Descriptor Table»), della quale parleremo nella prossima puntata, fa riferimento ad un task gate piuttosto che ad un segmento di codice per cui il selector del nuovo TSS descriptor relativo al task da attivare si troverà ancora una volta all'interno del task gate.

Come dunque abbiamo già più volte detto, per effettuare un task switch non è stata introdotta alcuna nuova istruzione, ma gioca un ruolo importante il tipo segment descriptor puntato dal selector presente nelle istruzioni CALL FAR e JMP FAR: anche se ne parleremo la prossima puntata, accenniamo al fatto che per l'istruzione INT vale un concetto analogo.

Infatti gioca un ruolo fondamentale il selector posto come parte più significativa dell'interrupt vector relativo all'interrupt («software» in questo caso) attivato dall'istruzione INT.

Altra possibilità è data anche dall'istruzione IRET, nel qual caso gioca un ruolo importante il già citato bit «NT» del flag.

Nel caso dunque di attivazione di un task per mezzo di un'istruzione di CALL FAR o di JMP FAR scatta l'oramai ben noto meccanismo di controllo dei livelli di privilegio ed in particolare, chiamando con EPL l'«Effective Privilege Level» e cioè il massimo tra i valori numerici dell'RPL («Requested Privilege Level») posto all'interno del selector e del CPL («Current Privilege Level») del task in corso di esecuzione, vale la regola che l'EPL deve essere minore o uguale al valore del DPL («Descriptor Privilege Level») proprio del descrittore del TSS, posto come noto nell'«Access Rights Byte» del descrittore stesso.

Sembrano inevitabili giochi di parole (tra EPL, RPL, CPL e DPL), ma invece una volta capito il meccanismo il resto viene avanti da solo: la regola quasi lapalissiana è ancora una volta che per poter compiere un'azione qualsiasi bisogna averne il privilegio e cioè il livello di privilegio di partenza deve essere sempre minore o uguale (numericamente!!!) del livello di privilegio dell'«oggetto a cui miriamo». E che questi livelli di privilegio abbiano fino a quattro possibili nomi non è certo per una eccessiva necessità di formalismo da parte dei progettisti.

Supponendo dunque che i livelli di privilegio consentano l'accesso al TSS (altrimenti si ha la solita segnalazione di errore di violazione di privilegi), il task switch avviene in cinque passi successivi, eseguiti in sequenza a meno che non si abbia nel frattempo una segnalazione di errore di qualsiasi genere, che fatalmente interromperà il task switch.

I cinque passi sono dunque i seguenti:

— il primo passo consiste nel testare se il task ha la possibilità di «switchare» verso un altro task, secondo le ben note regole che coinvolgono il privilegio di accesso ai dati appartenenti al nuovo task e con relative segnalazioni di errore in caso negativo.

— Nel secondo passo viene testata la presenza (bit P uguale ad 1) del nuovo task e la correttezza del campo LIMIT all'interno del descrittore del TSS: eventuali errori in questo punto vengono ovviamente gestiti nel contesto del task ancora in corso di esecuzione, in quanto il nuovo task non è ancora «noto».

— Nel terzo passo viene salvato lo stato del task in corso di esecuzione, che dunque diventa il «task uscente», all'interno del proprio TSS (ricordiamo che il selector al descrittore del TSS è posto nel registro della CPU chiamato TR, «Task Register»): in particolare viene riempita, in modo automatico dall'80286 e senza intervento di istruzioni particolari, la parte del TSS che abbiamo designata come «dinamica», relativa al contenuto di tutti i registri della CPU.

— Nel quarto punto viene a tutti gli effetti «marcato» come attivo il task entrante, caricando il registro TR con il selector al TSS relativo, settando ad uno il bit «B» («Busy») posto nel descrittore del TSS e settando il flag TS («Task Switched») all'interno della «Machine Status Word» (MSW), della quale parleremo in seguito: diciamo adesso che tale flag serve ad indicare all'eventuale coprocessore matematico che il contesto su cui l'80287 lavorava è stato cambiato.

— Nel quinto ed ultimo passo vengono caricati i registri della CPU a partire dai contenuti posti nel TSS del task «neonato»; in particolare viene caricato il registro LDTR («Local Descriptor Table Register») e viene effettuata una interminabile serie di controlli al termine della quale (in caso di esiti favorevoli) il task può finalmente andare in esecuzione. Errori a questo livello vengono ovviamente imputati al task «neonato», del quale perciò non verrebbe eseguita nemmeno la prima istruzione.

Abbiamo parlato di una lunga serie di controlli: per la cronaca è la seguente:

— si testa innanzitutto se il selector della LDT contenuto all'interno della TSS (del task entrante) è corretto e cioè se i limiti sono a posto e se l'LDT in questione è presente in memoria.

— Si controlla la validità del selector contenuto nel CS (salvato sempre all'interno del TSS), la presenza del relativo segmento (che deve essere un segmento di codice), nonché il livello di privilegio del descrittore del segmento (DPL) rispetto all'RPL contenuto nel CS.

— Si testa lo stack segment e cioè se è valido, se è un segmento di dati in cui si possa accedere in scrittura, se è presente e se al solito il DPL del descrittore del segmento è uguale al CPL.

— Si testa se i selettori contenuti nei registri DS ed SS sono corretti, se i segmenti a cui si riferiscono sono di tipo «readable», se i segmenti in questione sono effettivamente presenti in memoria ed infine se (è quasi inutile dirlo...) i livelli di privilegio siano ad hoc.

Il minimo errore a questo punto, come detto, rende impossibile l'attivazione del task e si può vedere che già a partire dal terzo passo viene salvato lo stato del task uscente, che perciò potrà poi essere riattivato correttamente, come se niente fosse successo.

Con questo terminiamo questa puntata alquanto più breve del solito, ma ugualmente ricca di terminologie che oramai speriamo non siano più tanto oscure, visto che prima o poi le si ritrovano quando meno le si aspettano.

Nella prossima puntata inizieremo ad analizzare il comportamento dell'80286 in risposta ad un interrupt: alcune caratteristiche ricorderanno alla lontana il funzionamento del capostipite della famiglia Intel, l'8086, ma per altri versi ritroveremo altre strutture particolari, simili però a quelle viste finora. MC