di Pierluigi Panunzi

Una nuova struttura: il «gate»

Dopo aver parlato di particolari strutture richieste dall'80286 nel suo funzionamento in modo protetto, continuiamo imperterriti nell'analisi di altre: sembra quasi che non finiscano mai...
Dopo i «Table Descriptor» con i loro annessi e connessi, dopo l'analisi dei «Segment Descriptor» e dei livelli di privilegio ecco che tocca ora ad un'altra struttura denominata «gate», che gioca un ruolo determinante in quanto permette di regolare i trasferimenti di controllo da un segmento ad un altro, verificando da un lato che non si vada contro la protezione di memoria e dall'altro contro livelli di privilegio

I trasferimenti di controllo: salti e chiamate a subroutine

Sappiamo molto bene che tra i registri di segmento, il CS (Code Segment) gode della particolarità di non poter essere caricato direttamente con un'istruzione, cosa che invece è possibile ed usata molto spesso con i vari DS, ES ed SS.

È infatti chiaro che il CS non può essere cambiato così senza tante precauzioni in quanto è lui che contiene istante per istante, assieme al registro IP (Instruction Pointer), l'informazione di quale istruzione deve essere eseguita; se fosse possibile caricare il CS per mezzo di un'istruzione del tipo

MOV CS,ALFA

ecco che difficilmente la prossima istruzione da eseguire potrebbe essere quella posta subito dopo la MOV (a meno che in ALFA non ci sia proprio il valore del segmento attuale...).

Per fortuna dunque non è possibile caricare direttamente il CS (e sarebbe stata una gioia per i programmatori dediti al "HaraKiri"...), mentre viceversa ciò può ottenersi (ed anche questo è ben noto ed ovvio, ma non si sa mai...) per mezzo di una istruzione di «JMP inter-segment» oppure una «CALL inter-segment» tanto direttamente quanto indirettamente, le quali istruzioni in un sol colpo cambiano il valore del registro CS nonché dell'IP.

Sappiamo che le istruzioni «dirette» contengono proprio in esse i valori da porre nei due registri citati, mentre quelle indirette si appoggiano necessariamente a due word poste in memoria: a parte il meccanismo differente di provenienza dei due valori ed i relativi differenti tempi di esecuzione, il risultato finale è sempre e comunque quello di trasferire il controllo ad un altro punto della memoria di programma.

Tutto questo filava liscio come l'olio per l'8086 e per il 286 in modo reale: per quanto riguarda il modo protetto, sappiamo ancora meglio che una semplice operazione come quella citata viceversa coinvolge parecchie risorse della CPU ed in particolar modo gli «schemi di protezione» (che sa tanto di fantascientifico...) nonché i «livelli di privilegio» attraverso descrittori di segmenti, byte di diritti di accesso e tabelle locali e/o globali.

In aggiunta a tutto questo ecco apparire una nuova struttura che rappresenta una specie di «filtro», o «guardiano» o «cancello» (infatti si chiama proprio «gate») che, a discrezione di come è stato «comandato» può inibire o permettere il passaggio di controllo da un segmento all'altro specie in presenza di salti di livello di privilegio.

Rassicuriamo ancora una volta i lettori che, come quasi tutte le strutture fin qui incontrate, anche il «gate» risulterà trasparente al programmatore, tanto che non ci si accorgerà della sua presenza: al solito il compito maggiore è svolto dal «programmatore di sistema» e dal sistema operativo che viceversa deve creare e tenere sotto controllo istante per istante tutte queste strutture particolarmente delicate: una notevole mano al sistema operativo è ovviamente data dalla CPU che è stata creata proprio per questi compiti e non si scompone minimamente...

Dicevamo dunque dei salti inter-segment: in particolare ora ci interessa la distinzione tra salti a segmenti di pari privilegio e i salti a segmenti di differente livello. Una cosa importantissima da sottolineare è che mentre i salti inter-segment di pari privilegio possono essere effettuati da istruzioni di «JMP», «CALL» e «RET» (RET di tipo «far»), al contrario i passaggi tra segmenti a differente privilegio possono solamente avvenire per mezzo di «CALL» e di «RET», ma non con «jump»: ecco che perciò è chiaro il fatto che il passaggio ad un segmento di livello di privilegio differente deve essere, come dire, «fugace», quale quello ottenibile con una chiamata ad una subroutine alla quale si salta per poi ritornare con una RET.

Vietata invece è la situazione che porterebbe un processo a sconfinare e stabilirsi in un segmento di privilegio differente, dal quale non ci si allontanerebbe più, il tutto a causa di una JMP che viceversa per sua natura non prelude ad alcuna RET.

Fatta questa precisazione, si ha che i

salti inter-segment verso segmenti di pari privilegio ed anche quelli all'interno dello stesso segmento (fatto che non altera i privilegi) vengono controllati e verificati con tutti quei meccanismi visti finora, mentre appunto laddove c'è una differenza di privilegi si innesca un nuovo meccanismo, che deve salvaguardare l'integrità del sistema.

In particolare prima di poter accedere ad una certa routine posta ad un livello di privilegio differente, la parola passerà ad un nuovo meccanismo che controllerà innanzitutto che al task corrente sia permesso l'accesso alla routine desiderata e successivamente (e questa è un'altra novità) che l'indirizzo della routine (l'«entry point») sia quello corretto: a questo serve dunque la struttura chiamata «gate».

Il nostro task potrà dunque eseguire routine e programmi a differenti livelli di privilegio senza quasi accorgersi che nei casi migliori il controllo è passato direttamente, mentre nei casi peggiori il controllo passa attraverso un «gate».

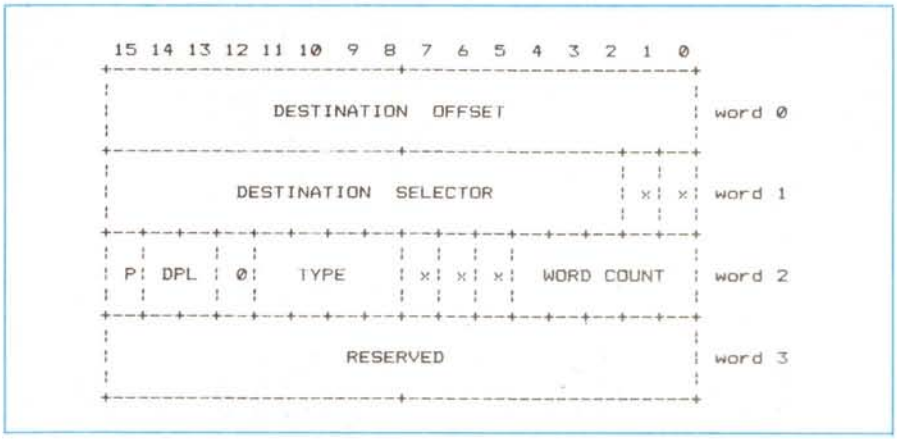

Figura 1 - La struttura di un generico «gate»: per il significato dei suoi campi rimandiamo al testo.

Com'è fatto un «gate»

Così come altre strutture già viste in precedenza, un gate è formato da quattro word: in particolare assomiglia un poco ad un segment descriptor ed in realtà serve per rappresentare ognuno di quattro tipi di «gate» che conosceremo successivamente.

I quattro tipi di gate che l'80286 prevede, e che serviranno a permettere o meno una certa operazione o funzione, sono

— il «call gate» (il cui nome fa subito e fatalmente pensare ad un ben noto dentifricio...)
— il «task gate»
— l'«interrupt gate»
— il «trap gate».

Dal momento che abbiamo parlato finora di chiamate a subroutine a livelli di privilegio differente, ecco che dovrebbe essere facile arguire che nel nostro caso si tratterà di un «call gate»: la sua struttura è riportata in figura 1, laddove vediamo che si differenzia dagli altri «gate» per il contenuto del campo «TYPE».

Ma prima di analizzare in dettaglio la figura, andiamo a conoscere meglio la funzione svolta da un gate, indipendentemente dal suo tipo (call, task, interrupt o trap gate): tutti e quattro i gate definiscono un indirizzo al quale verrà passato il controllo, indirizzo che altrimenti in generale non potrà essere direttamente raggiunto da programma.

Si tratta dunque di una sorta di «chiave di accesso» ad una certa routine della quale non si conosce l'indirizzo (che potrà magari cambiare), ma alla quale si può fare riferimento tramite il gate: un po' come i ben noti «interrupt vector» presenti nell'8086 (ed ovviamente ancora ben funzionanti nell'80286) che servono ad indirizzare una certa routine di gestione di interrupt a partire dal valore dell'interrupt generato, che veniva dunque trattato come selettore.

Nel nostro caso il selettore è proprio il «selector» contenuto all'interno dell'istruzione di salto oppure nella locazione di memoria nel caso di salto indiretto, selector che dovrebbe essere caricato nel registro CS: invece, mentre la parte di «offset» dell'indirizzo della routine viene ignorata, la parte relativa al «selector» serve ad identificare un determinato «gate» dal quale si potrà successivamente estrarre l'entry-point della routine.

Un vantaggio notevole della struttura «gate» è che la sua allocazione in memoria rimane sempre fissa (mentre viceversa il suo contenuto può variare), ottenendo in tal modo che per accedere ad una routine facciamo riferimento comunque ad un indirizzo fisso, anche se la routine vera e propria potrà avere nel tempo indirizzi differenti, ai quali, lo ripetiamo, è difficile (anzi impossibile) fare riferimento direttamente.

Ciò aiuta inoltre a spiegare il perché non è altrettanto lecito effettuare uno «jump» ad una routine di differente livello (in generale una routine di Sistema Operativo) o peggio in un punto all'interno di essa.

Perciò a seconda del tipo di gate e della situazione si avrà un differente comportamento di tutta la CPU e conseguentemente del programma: nelle prossime puntate conosceremo gli altri tipi di gate, mentre ora parleremo diffusamente del «call gate».

Il «call gate»

Facendo dunque riferimento alla figura 1, riconosciamo ancora una volta una struttura abbastanza familiare, nella quale le prime due word sono in un qualche modo legate ad un indirizzo, mentre nella terza word appare qualcosa di simile ad un «Access Rights By-

te». Vediamo dunque prima quali sono le funzioni svolte da un «call gate», per poi confrontarle con il contenuto della struttura.

Abbiamo detto che tale tipo di gate è richiesto da istruzioni di «CALL» e di «JMP», esattamente come se si trattasse di un «segment descriptor»: in particolare all'atto dell'esecuzione dell'istruzione che prevede un aggiornamento della coppia CS:IP, sappiamo che il «selector» va a selezionare un elemento della tabella LDT o GDT, secondo quanto abbiamo già analizzato ed elucubrato in puntate precedenti (e che ora non ripetiamo).

Ora, se il selector è diverso da 0 e se non si fuoriesce dalla tabella dei descrittori (in caso contrario viene generato un errore), allora al posto di un generico «segment descriptor» dovremmo trovare proprio un «gate» (nel caso si tratti di un segment descriptor verrà generata la solita segnalazione d'errore).

Trovato in particolare un «call gate», allora si va a vedere se l'istruzione era una «JMP» o una «CALL»: nel primo caso viene concesso il salto solo se i livelli di privilegio sono identici, altrimenti viene generata una segnalazione di errore, mentre se si trattava di una «CALL», allora indifferentemente dalla differenza dei livelli di privilegio viene effettuato l'accesso alla routine.

Se la routine non è effettivamente presente (ricordate il bit di presenza «P» di un segment descriptor? Anche nel nostro caso esiste...) allora viene generata un'altra segnalazione che probabilmente verrà utilizzata dal Sistema Operativo stesso per caricare in memoria fisica (da quella virtuale di massa) la routine che ci interessa.

Tutto ciò, lo ripetiamo, risulta del tutto trasparente per il programma e per il programmatore.

Subito prima però di concedere definitivamente l'accesso alla routine, la CPU effettua altri passi intermedi consistenti appunto nel determinare l'indirizzo di trasferimento e se è richiesta una transizione di livello di privilegio (importantissima per poter accedere ai dati di sistema da parte di routine applicative): tale transizione di livello di privilegio, nel caso ciò sia richiesto, viene così effettuata, così come viene effettuato (era ora!) il salto alla routine.

Siamo ora dunque in grado di analizzare nel dettaglio la figura 1, nella quale troviamo vari campi:

— il campo DESTINATION OFFSET insieme al campo DESTINATION SELECTOR rappresentano l'indirizzo...

No! Sembrava così facile! Andiamo per gradi:

— il DESTINATION SELECTOR è ancora una volta il selettore del segmento in cui si trova la nostra routine e perciò fa riferimento ad un elemento (segment descriptor) all'interno della GDT o LDT e come tale determinerà la possibilità di accesso alla routine in base alla possibilità di accesso al segmento in cui risiede la routine stessa.

Sembra un circolo vizioso, ma invece è sempre lo stesso diabolico meccanismo che si ripete in ogni occasione!!

— Il campo DESTINATION OFFSET stavolta è proprio l'offset della routine all'interno del suo segmento di appartenenza: in parole povere è l'offset dell'«entry point» della routine stessa;

— il campo P indica la presenza o meno del descrittore e perciò la validità o meno dei dati eventualmente contenuti nei vari campi;

— il campo DPL è il solito «Descriptor Privilege Level»;

— il campo che vale «0» è quello che differenzia un «gate» da un «segment descriptor», che invece ha tale bit ad 1;

— il campo TYPE è quello che differenzia i vari tipi di gate ed in particolare si hanno i seguenti valori:

TYPE	TIPO DI GATE
100	call gate
101	task gate
110	interrupt gate
111	trap gate

— Il campo WORD COUNT indica, con

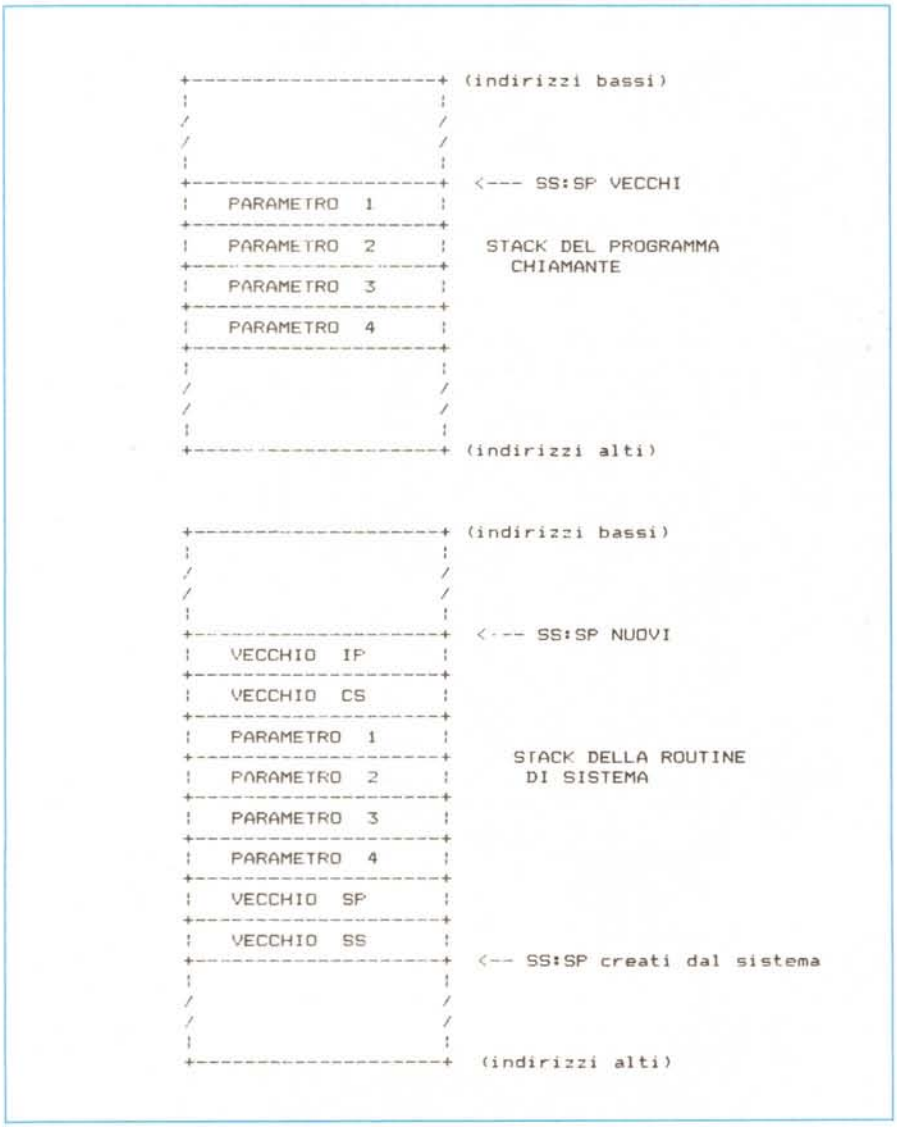

Figura 2 - Rappresentazione schematica del «doppio stack» che viene creato allorché da un processo a basso privilegio (un programma applicativo) si passa il controllo ad una subroutine ad alto livello di privilegio (subroutine di sistema).

valori compresi tra 0 e 31, il numero di word da copiare dallo stack del programma chiamante sullo stack della procedura chiamata. Ne riparleremo fra breve;
— il campo RESERVE è ancora una volta riservato e va posto al solito a 0 per compatibilità futura con l'80386, che viceversa userà tale campo.

Parlavamo dunque di possibilità che il salto avvenga a parità di livello di privilegio oppure a differente privilegio.

Nel caso di uguaglianza di livelli l'accesso è subordinato a quanto detto precedentemente laddove aggiungiamo che comunque viene testato se l'offset a cui dobbiamo saltare si mantiene all'interno del segmento desiderato, altrimenti si ha una segnalazione opportuna di errore.

Il «call gate» in questo caso rappresenta un passaggio in più nella generazione dell'indirizzo da porre in CS:IP, che non risentirà, come già sottolineato, del fatto che la routine chiamata può trovarsi in posti differenti della memoria in istanti differenti: il «call gate» serve appunto per sapere in ogni istante l'indirizzo della routine.

Chiamata a subroutine con differente livello di privilegio

Nel caso di differenti livelli di privilegio il «call gate» rappresenta un elemento necessario e sufficiente per l'effettuazione della chiamata alla subroutine.

Infatti è una garanzia per il Sistema Operativo dal momento che la chiamata ad una routine ad alto privilegio da parte di un applicativo a basso privilegio avviene ad un indirizzo sicuramente corretto, piuttosto che magari nel bel mezzo della routine stessa, mentre viceversa risulta obbligatorio passare attraverso un gate per accedere a routine a livello di privilegio maggiore.

Dal momento che si tratta di una chiamata a subroutine sappiamo già che si dovrà salvare nello stack il valore corrente del CS e dell'IP: in particolare il campo RPL all'interno del selector del CS sarà proprio il valore del livello di privilegio del programma chiamante, che servirà poi al momento del ritorno dalla subroutine per ripristinare la situazione, così come era subito prima della chiamata alla subroutine.

Un altro problema: lo stack

Lavorando con subroutine ad alto livello di privilegio, sorge il nuovo problema dello stack, sul quale abbiamo eventualmente lavorato con il programma applicativo, per mezzo del quale possiamo passare eventuali parametri ed infine sul quale lavorerà la subroutine ad alto livello di privilegio.

Per motivi di sicurezza e di integrità del sistema queste tre gestioni dello stack non possono avvenire fisicamente in un'unica zona di memoria in quanto con i soliti trucchi programmativi si potrebbero bypassare i controlli del 286.

Ecco che perciò, all'atto della chiamata di una subroutine ad alto privilegio, è prevista la creazione di un nuovo stack a privilegio maggiore sul quale lavorerà la subroutine: il problema nasce da come fare a passare eventuali parametri alla subroutine, oltre all'ovvio indirizzo di ritorno.

A questo scopo serve appunto il campo WORD COUNT all'interno di un «call gate», che indica quante word (da nessuna fino ad un massimo di 31) devono essere copiate dallo stack del processo chiamante allo stack della subroutine di sistema: queste word saranno gli eventuali parametri, il vecchio valore della coppia CS:IP (che punta proprio all'istruzione successiva a quella di chiamata alla subroutine, all'interno del nostro programma), nonché il valore vecchio della coppia SS:SP che punta invece allo stack «vecchio» del programma, chiamante.

In particolare il salvataggio di SS:SP serve tra l'altro in quei casi in cui le word da copiare da uno stack all'altro sono più di 31 ed in tal caso la subroutine potrà accedere alle word in più proprio grazie ai vecchi SS:SP, andandole a ricercare nello stack del programma chiamante: complicato ma efficace...

Per chiarire un attimo la situazione facciamo riferimento alla figura 2, dove vediamo lo stato dei due stack, uno del programma chiamante, che richiede 4 parametri e l'altro della subroutine di sistema.

Il Sistema Operativo allocherà perciò un nuovo stack ad alto livello di privilegio (puntato da «SS:SP creati dal sistema») dove verranno dapprima salvati i valori di SS e SP del programma chiamante (ricordiamo che lo stack cresce verso indirizzi bassi e cioè «verso l'alto» avendo rappresentato correttamente la memoria con indirizzi bassi sopra ed indirizzi alti sotto, a differenza dei diagrammi della casa madre Intel che sono sempre rovesciati), poi i quattro parametri ed infine l'indirizzo di ritorno.

Cosa succede al «ritorno»

Ad un certo punto la subroutine ad alto livello di privilegio giunge al termine, allorquando dovrà eseguire generalmente una «RET nn» e cioè, lo ricordiamo, una RET in cui è indicato il numero di word che devono essere scartate dallo stack prima di cedere il controllo al programma che si trova all'«indirizzo di ritorno» (proprio la prosecuzione del nostro programma!).

In particolare il ritorno può avvenire solo da una subroutine ad alto privilegio verso un programma chiamante a basso privilegio (è proprio il nostro caso), mentre il viceversa scatena un'opportuna segnalazione d'errore.

Dopo questo controllo dei livelli di privilegio, vengono ripristinati i vecchi CS ed IP (a patto che, al solito, CS non sia nullo, che il «segment descriptor» relativo sia all'interno della relativa «descriptor table», che il segmento puntato sia proprio un segmento di codice e che il segmento sia effettivamente presente in memoria) ed inoltre viene aggiornata la coppia SS:SP ai valori di prima della chiamata, corretti in modo da scartare i parametri non più voluti.

In particolare ripristinando il CS, automaticamente si avrà un riaggiustamento del livello di privilegio del programma chiamante: a conclusione delle operazioni, viene effettuato un ulteriore controllo su DS ed ES, verificando che entrambi puntino a segmenti aventi il privilegio uguale a quello del nostro programma chiamante (la sicurezza innanzi tutto...) in modo da evitare la minima possibilità da parte di un programma ad infimo livello di privilegio di sfiorare dati di sistema appartenenti ed usate da routine di Sistema Operativo a supremo livello di privilegio.

Nel caso in cui viceversa il DS e/o l'ES puntino laddove non è lecito, allora vengono brutalmente inizializzati ad un valore nullo, fatto che lì per lì non comporta alcune conseguenze: però il primo tentativo di accesso a locazioni di memoria contenenti dati da parte del nostro programma (che si credeva così di accedere a dati di sistema) viene frustrato con un'opportuna segnalazione d'errore dovuta appunto al valore nullo di un segment selector, cosa che già sappiamo essere illecita.

Tra parentesi finora abbiamo sempre parlato genericamente di «opportune segnalazioni d'errore» senza specificare cosa succede in realtà: ne riparleremo a tempo debito e vedremo che molto dipenderà da come è implementato il sistema operativo.

Con questo dunque il controllo è tornato al nostro processo, per cui siamo felici e contenti...