Modo reale e modo protetto

Una delle caratteristiche più interessanti del Borland Pascal 7.0 è la possibilità di compilare programmi eseguibili in modo protetto, secondo le specifiche DPMI: se il computer è dotato di un processore 80286 o superiore, si ottiene così accesso a tutta la memoria estesa disponibile, fino a 16 megabyte. È necessario che i file DPMI16BI.OVL (server DPMI) e RTM.EXE (runtime manager) siano installati nella stessa directory dell'applicazione o in una indicata nella variabile PATH dell'environment, in quanto si richiede un DOS extender. Anche Windows 3.x è però, a suo modo, un DOS extender; è possibile, quindi, approntare applicazioni che, eseguite in una sessione DOS di Windows, siano in grado di superare la tradizionale barriera dei 640K. Si possono anche usare librerie dinamiche (DLL), comprese quelle di Windows. In sintesi, un nuovo e ricco scenario che merita la nostra attenzione

di Sergio Polini

Tutti ben sappiamo che, di norma, un'applicazione eseguita sotto DOS non può disporre di più di 640K di memoria per il codice e i dati. Sappiamo anche che tutto ciò dipende dal modo in cui i processori 80x86 costruiscono un indirizzo fisico: si moltiplica per 16 il valore del segmento e gli si somma il valore dell'offset; si ottiene così un numero di 20 bit, attraverso il quale possono essere espressi indirizzi variabili da 0 a 1.048.575, compresi, quindi, nell'ambito di un megabyte di memoria. Tale spazio viene suddiviso in un'area utente (i primi 640K) e in un'area di sistema (i restanti 384K), riservata principalmente al BIOS ed alla memoria video.

Questo avviene nel cosiddetto *modo reale*, in cui segmento e offset esprimono, appunto, le due componenti di un indirizzo *fisico*: moltiplicando per 16 il segmento e aggiungendo l'offset si ottiene un numero che esprime direttamente una locazione nella memoria installata nel computer.

Con l'80286, e soprattutto con i successivi 80386 e 80486, a quello che era l'unico "modo" disponibile si affianca una diversa tecnica di indirizzamento; il segmento, da mera componente di un indirizzo fisico, si trasforma in indice in una tabella di *descrittori* che contengono, oltre all'indirizzo dell'inizio di un segmento di memoria, anche informazioni sulle sue caratteristiche; in particolare, ad ogni segmento di memoria vengono attribuiti sia una dimensione che diritti d'accesso, rendendo così possibile il riconoscimento, ad opera del processore, di tentativi di accedere a locazioni di memoria poste oltre il limite del segmento o di modificare per errore, come se si trattasse di un'area dati, un segmento contenente codice eseguibile, oppure di leggere o modificare le zone di memoria riservate al sistema operativo. Per questo motivo, si parla di un *modo protetto* distinto dal tradizionale modo reale; la combinazione dell'indirizzo contenuto nel descrittore e di un offset, inoltre, conduce ad un indirizzo a 32 bit interpretabile come indirizzo fisico in uno spazio di 4 gigabyte sia direttamente, sia attraverso un meccanismo di paginazione mediante il quale è possibile implementare sistemi di memoria virtuale, in cui la memoria disponibile è superiore a quella fisicamente installata grazie all'uso del disco come area di parcheggio di blocchi di memoria temporaneamente non utilizzati.

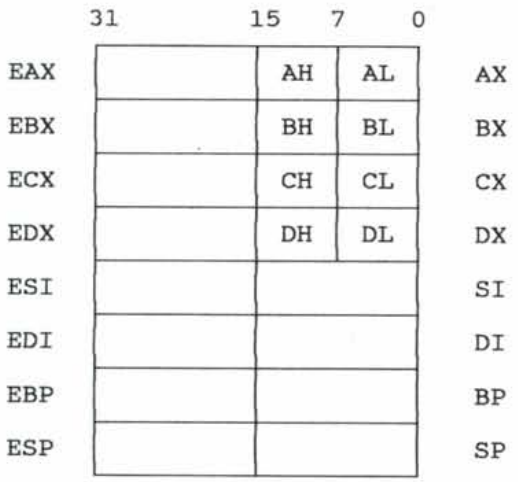

Figura 1 - I registri di uso generale a 32 bit, comprendenti i "vecchi" registri a 16 bit (ad esempio, i primi 16 bit del registro EAX corrispondono al tradizionale registro AX, scomponibile in AH e AL).

I segmenti come selettori

Per illustrare sinteticamente la tecnica di indirizzamento in vigore nel modo protetto, sarà bene prescindere dalle

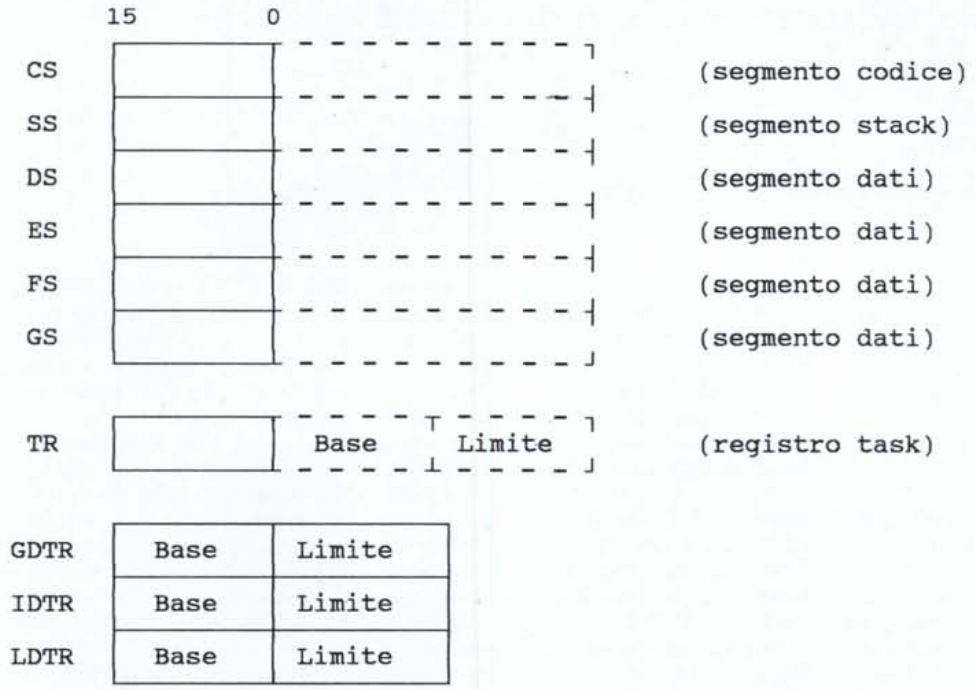

Figura 2 - I registri di segmento, quasi tutti comprendenti una parte "visibile" (selettore), corrispondente ai tradizionali registri di segmento, e una parte "invisibile" (descrittore), cui ha accesso solo il processore; fanno eccezione i registri TR, GDTR, LDTR e IDTR.

differenze tra i vari processori, concentrandoci sull'80386.

In esso troviamo 34 registri: 8 registri di uso generale (figura 1), 10 registri di segmento (figura 2), 6 registri di controllo (figura 3) e 10 registri per debug e test (su cui non ci soffermeremo). Oltre ai registri, vi sono "oggetti di sistema" quali descrittori, tabelle di descrittori, segmenti e porte (*gate*). Ogni segmento o *gate* ha un descrittore di 8 byte che lo definisce. L'accesso alla memoria avviene mediante un descrittore.

Il formato dei descrittori è illustrato nella figura 4. Il processore ricompone i tre campi BASE per costruire l'indirizzo a 32 bit dell'inizio di un segmento; i due campi LIMITE definiscono la dimensione del segmento mediante un valore a 20 bit che può essere interpretato in due modi: se il bit di granularità (G) è 0, si usano unità di 1 byte, con un massimo di un megabyte, se quel bit è 1, si usano unità di 4 Kbyte, con un massimo di 4 gigabyte. Il bit D viene detto *default bit* e indica se si usano il set di istruzioni e i modi di indirizzamento a 32 bit, o se si devono interpretare le istruzioni come se il processore fosse un 80286. Il bit AVL è disponibile per la programmazione di sistema; i bit G, D, AVL e il bit nullo compreso tra questi ultimi, unitamente al campo LIMITE 16..19, vengono collettivamente detti *byte di granularità*. Viene invece detto *byte di accesso* l'insieme dei campi P, DPL, TIPO e A; il primo è un bit che indica se il segmento è presente in memoria (potrebbe essere stato "parcheggiato" su disco), il campo DPL indica il livello di privilegio (da 0 a 3), il campo TIPO indica il tipo del segmento, il bit A viene settato quando si accede al segmento, cioè tipicamente quando in un registro di segmento viene caricato il selettore per il descrittore di quel segmento.

Vi sono diversi tipi di segmenti; nell'uso normale, si distingue soprattutto tra segmenti solo "leggibili" (codice, dati costanti) o anche "scrivibili" (variabili). Un segmento per codice eseguibile avrà tipo 11CR, essendo R il bit che indica la leggibilità di segmento e C il cosiddetto *conforming bit*, settato per segmenti contenenti codice che deve essere eseguito allo stesso livello di pri-

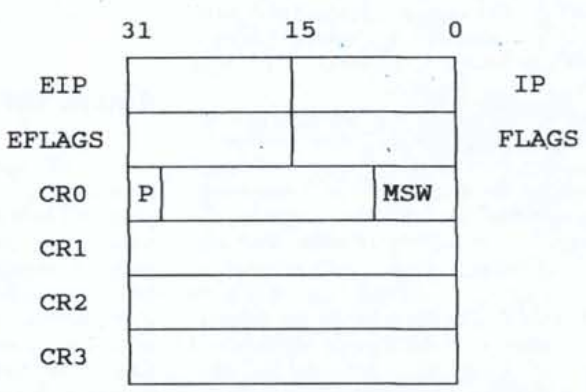

Figura 3 - I registri di controllo, alcuni interpretabili come estensioni a 32 bit dei vecchi registri di controllo dell'8086, altri nuovi; CR0 comprende, peraltro, quello che era chiamato Machine Status Word nell'80286, il cui primo bit vale 1 se si è in modo protetto.

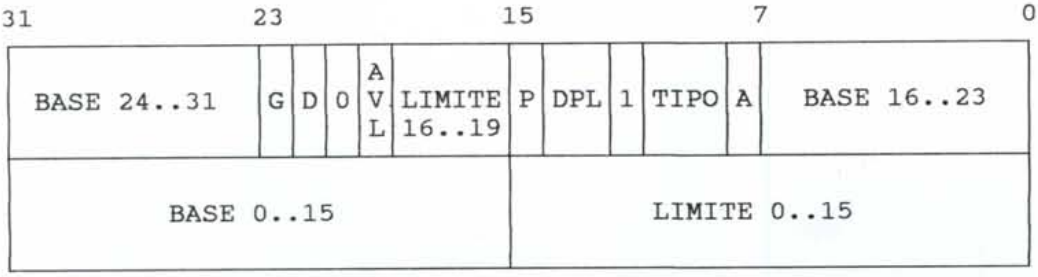

Figura 4 - Il formato dei descrittori usati per segmenti codice e dati di applicazioni. Vi sono altri formati per speciali segmenti di sistema e per i gate.

vilegio del codice da cui riceve il controllo (mediante una chiamata di procedura). Un segmento per i dati avrà tipo 10EW, essendo W il bit che indica la possibilità di variare i valori in esso memorizzati ed E un bit che indica se il segmento si espande verso l'alto (caso normale) o verso il basso (utilizzabile per lo stack; in questo caso l'interpretazione del campo LIMITE è più articolata di quanto sopra brevemente accennato).

Non ci soffermeremo sui livelli di privilegio; sarà sufficiente accennare alla possibilità di assegnare privilegio massimo (il livello 0) al sistema operativo e minimo (il livello 3) alle applicazioni e, quindi, di escludere l'accesso a codice e dati di sistema da parte dei programmi applicativi.

Tornando all'indirizzamento, rileviamo che i registri di segmento non vengono interpretati come locazioni di blocchi di memoria, ma come indici in tabelle di descrittori. Ogni tabella non è altro che un array di descrittori (con un massimo di 8192) il cui primo elemento, con indice zero, non viene usato. Vi devono essere almeno una tabella globale di descrittori (GDT) ed una tabella di descrittori di interrupt (IDT), accessibili, rispettivamente, mediante i registri GDTR e IDTR e coppie di istruzioni di LOAD e STORE (LGDT e SGDT, LIDT e SIDT). Sono possibili anche tabelle locali (LDT), utili per il multitasking e accessibili mediante il registro LDTR e le istruzioni LLDT e SLDT.

Quando si vuole accedere ad una locazione di memoria, l'indirizzo viene espresso mediante un registro di segmento ed un offset, su cui il processore opera in due fasi. In primo luogo, la parte visibile del registro di segmento viene interpretata come selettore: i bit da 4 a 15 vengono assunti come indice nella tabella globale o locale dei descrittori, secondo il valore del terzo bit (0 per GDT, 1 per LDT; i primi tre bit, da 0 a 2, indicano il livello di privilegio). Si accede così ad un descrittore, che viene caricato nella parte invisibile del registro di segmento (per evitare ripetuti accessi alla tabella) e che contiene, come abbiamo visto, l'indirizzo della base di un segmento di memoria e la sua dimensione; l'indirizzo cui si vuole accedere viene ottenuto aggiungendo l'offset alla base del segmento. Nella seconda fase, si verifica se è abilitato o meno il meccanismo di paginazione; se non lo è (se il bit più alto del registro CR0 vale zero), l'indirizzo così ottenuto, detto *indirizzo lineare*, viene interpretato come un indirizzo fisico a 32 bit, altrimenti i bit da 22 a 31 sono un indice in una directory di pagine (il cui indirizzo è mantenuto nel registro CR3) dalla quale si ottiene l'indirizzo di una tabella di pagine; i bit da 12 a 21 dell'indirizzo lineare costituiscono un indice in tale tabella, da cui viene così letta la base di una pagina, l'offset nella quale viene tratto dai bit 0 a 11 dell'indirizzo (figura 5).

Ogni pagina altro non è che un blocco di memoria di 4 Kbyte. Anche la directory e le tabelle di pagine sono a loro volta pagine; la directory può quindi contenere fino a 1024 indirizzi a 32 bit di tabelle, ogni tabella può contenere fino a 1024 indirizzi di pagine. In totale, quindi, vi possono essere fino ad un mega di pagine e quindi, essendo ognuna di 4 Kbyte, la memoria complessivamente indirizzabile in questo modo ammonta a 4 gigabyte.

Il modo V86

L'80386 agevola la scrittura di sistemi operativi multitasking mediante una struttura di dati detta *Task State Segment* (TSS), che contiene tutte le informazioni necessarie per riprendere un task dopo aver ceduto il controllo ad un altro. Anche questo è un aspetto che non possiamo approfondire in questa sede; ci limiteremo, quindi, a sottolineare che, nell'ambito delle sue possibilità di multitasking, il processore ammette task detti V86 (*Virtual 8086*) che, pur essendo per il resto normali task in modo protetto, interpretano in modo tradizionale indirizzi espressi con coppie segmento-offset e intercettano alcune istruzioni "critiche".

Un task V86 si propone come una "macchina virtuale" in grado di ritagliare, nell'ambito della memoria indirizzabile dal processore, uno spazio di un mega dedicato all'esecuzione delle istruzioni dell'8086, ma in cui si dispone anche dei registri FS e GS, di istruzioni come ENTER e LEAVE, nonché della possibilità di usare operandi a 32 bit.

Si entra in un task V86 quando, passando ad un nuovo task, risulta settato il flag VM nel registro EFLAGS ad esso associato mediante il *Task State Segment*. Una volta passato al modo V86, il processore, pur continuando ad operare in modo protetto, si comporta per molti aspetti come se fosse un 8086, al punto che è possibile eseguire il DOS stesso come task V86. Ne abbiamo un esempio ogni volta che apriamo una sessione DOS in Windows.

Naturalmente ci sono difficoltà: il DOS, innanzitutto, può rimanere il sistema operativo sottostante (come, appunto, in Windows), con tutte le funzioni che mette a disposizione del programmatore tramite l'INT 21h o altri; il DOS eseguito in un task V86, inoltre, nulla sa dell'ambiente in cui si trova e, quindi, rischia di contendere ad altri DOS eseguiti come task V86 o al DOS sottostante l'utilizzo delle risorse della macchina.

Per evitare tali conflitti, l'80386, quando esegue un task V86, genera un'eccezione di protezione generale ogni volta che gli viene proposta un'istruzione CLI, STI, LOCK, PUSHF, POPF, INT o RET. I task V86, quindi, richiedono un *V86 monitor*, che è un normale programma 80386 eseguito in modo protetto che si incarica sia di inizializzare i task, sia di intercettare l'eccezione di protezione generale per verificare se la stessa è stata generata da una di quelle istruzioni, al fine di emular-

la e di mantenere così il pieno controllo delle operazioni.

I DOS Extender

Da quanto detto, appare evidente l'utilità di poter sfruttare appieno le caratteristiche del processore; è però anche necessario porre questo in modo protetto e, soprattutto, disporre di un ambiente che consenta la gestione trasparente di tutte le risorse della macchina, dalla tastiera alla memoria di massa. Il problema trova facile soluzione se si usa un sistema operativo appositamente scritto per l'80386; se, però, si deve o si vuole restare al DOS, occorre arrangiarsi.

Passare dal modo reale al modo protetto e viceversa non è difficile, come già sanno gli abbonati a MC-link; se ne è discusso recentemente nell'area ASM-80X86, e nell'area Programmi è disponibile un file PROTMODE.ZIP, contenente i sorgenti commentati di un programma in assembler che mostra in concreto come si deve procedere. Il vero problema è rappresentato dagli interrupt generati dalle periferiche e dalla necessità di usare i servizi del BIOS e del DOS.

A ciò soccorrono i cosiddetti *DOS Extender*, vere e proprie interfacce tra programmi eseguiti in modo protetto e un DOS che conosce solo il modo reale. Un DOS Extender, una sorta di *monitor* analogo a quello richiesto dal modo V86, intercetta le chiamate di interrupt quali il 10h, il 16h e il 21h, per poi gestirle direttamente o passarle al DOS con i dovuti accorgimenti. Un esempio può essere rappresentato dalla funzione 40h dell'INT 21h, mediante la quale si scrivono su disco tanti byte quanto è il valore del registro CX, con un massimo, quindi, di 64 Kbyte; un DOS Extender può consentire di chiamare quella funzione utilizzando appieno il registro ECX, con un massimo di 4 gigabyte, e lo farà traducendo quella richiesta in chiamate multiple della "vera" funzione del DOS. Nel caso di una richiesta di allocazione di memoria, al contrario, il DOS Extender provvederà direttamente, non potendo il DOS gestire la RAM oltre il primo megabyte.

L'esecuzione delle funzioni del DOS e del BIOS può avvenire in due modi. Alcuni DOS Extender riportano il processore al modo reale in occasione di ogni chiamata del DOS o del BIOS, per ripristinare il modo protetto subito dopo; un meccanismo facile da implementare, ma impraticabile se si vogliono sfruttare le capacità di multitasking del processore. Altri creano un task V86, cui viene passato il controllo in occasione di ogni interrupt, sfruttando così la possibilità di eseguire codice scritto per l'8086 senza abbandonare il modo protetto.

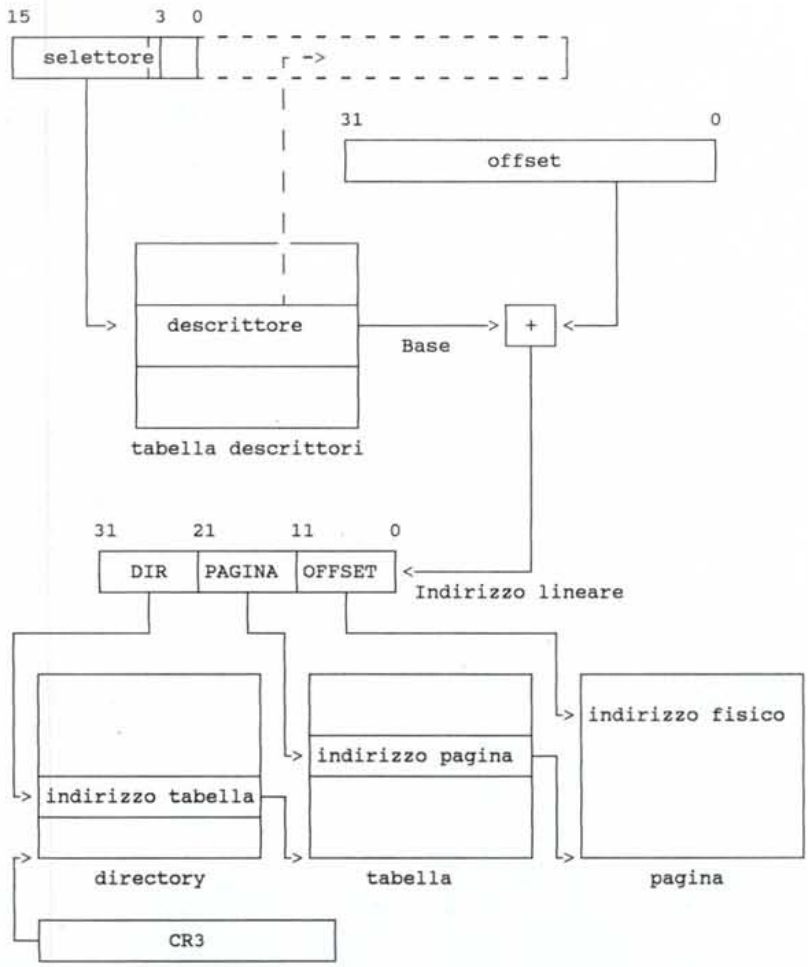

Figura 5 - Trasformazione di un indirizzo espresso mediante un registro di segmento ed un offset in un indirizzo lineare e sua interpretazione nel caso il meccanismo di paginazione sia abilitato.

Dal punto di vista del programma applicativo, la presenza del DOS Extender è in buona parte molto discreta; tuttavia, la necessità sia di comunicare in qualche modo con l'hardware, sia di adottare soluzioni specifiche per operazioni fuori della portata del DOS (quale l'allocazione di grandi quantità di memoria), impongono alcune regole di comportamento, formalizzate in apposite specifiche. Attualmente risultano dominanti le specifiche della versione 0.9 della *DOS Protected Mode Interface* (DPMI), comprendenti un sottoinsieme di chiamate del DOS e del BIOS effettuabili in modo protetto, nonché un INT 31h attraverso il quale allocare memoria, modificare descrittori, eseguire codice scritto per il modo reale, ecc.

Il mese prossimo, dopo una breve descrizione del supporto fornito dai file RTM.EXE e DPMI16BI.OVL, vedremo in concreto come un programma Pascal compilato per il modo protetto possa utilizzare sia le funzioni del DOS e del BIOS, sia quelle messe a disposizione dall'INT 31h. MC

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo MC1166@mclink.it.