di Pierluigi Panunzi

Alcune novità

La scorsa puntata abbiamo iniziato l'analisi delle prime caratteristiche del microprocessore 80386, partendo dai registri che in un certo senso non sono molto innovativi rispetto a quelli dei predecessori, a parte il fatto che sono tutti a 32 bit: d'altronde per compatibilità verso il basso non si possono avere nuove caratteristiche già a questo livello.
Le innovazioni però ci sono e pure parecchie...
Non rimane altro che analizzarle passo passo

Alcuni nuovi registri: i «Control Registers»

Tanto per dare un'occhiata alle novità, in quanto poi nel seguito approfondiremo gli argomenti che ora sarebbe troppo prematuro affrontare, il 386 possiede quattro registri interni a 32 bit (dei quali uno non utilizzabile, però), gestibili solamente a livello di sistema e cioè solo a privilegio più alto: si tratta dei quattro «Control Registers», che vanno da CR0 a CR3.

Diciamo subito che CR1 non ha alcun significato per il 386 e non deve essere utilizzato per non incorrere in incompatibilità verso l'alto e cioè (supponiamo) verso quell'80486 di cui da poco tempo si è cominciato a parlare tangibilmente: ancora una volta (e capiterà parecchie volte) i bit di tale registro sono da considerarsi «Intel Reserved».

Viceversa CR0, che vediamo in figura 1, rappresenta innanzitutto un'estensione verso i 32 bit del registro MSW («Machine Status Word») già presente nell'80286, che ne forma la parte meno significativa: della parte più significativa viceversa viene usato un solo bit, l'MSB, che si chiama PG («PaGing Enable»), ed avente il compito di abilitare oppure disabilitare l'unità di «paging» di memoria presente all'interno del chip.

Lo diciamo ora, ma poi non lo diremo più: «... di questo argomento parleremo in dettaglio in una delle prossime puntate...».

Gli altri 5 bit, dal più al meno significativo, invece, come detto, sono gli stessi che si trovavano nel 286 e cioè:

— ET («processor Extention Type») è un bit tuttofare che serve ad indicare, dopo il reset del 386, quale tipo di coprocessore è presente, se l'80287 o l'80387 (che tra l'altro è ovviamente a 32 bit), mentre da programma può essere settato per avere un protocollo di funzionamento a 32 bit (verso il 387) oppure (se resettato) a 16 bit per compatibilità verso il 287.

— TS («Task Switched») è il bit che viene settato automaticamente quando viene eseguita un'operazione di switching di task: anche in questo caso, come nella maggior parte delle situazioni, quanto sappiamo al riguardo per averlo visto nel 286 vale senz'altro come idea di base per il 386 a parte qualche differenza che ci premureremo di segnalare.

— EM («EMulate coprocessor») è un bit che può servire per far emulare il coprocessore laddove non sia fisicamente presente nel sistema: infatti se tale bit è settato, ad ogni istruzione riservata per il coprocessore viene generata un'eccezione (la numero 7, «Coprocessor Not Available Fault»), alla quale il «bravo sistema operativo» dovrebbe rispondere con apposite routine di emulazione, per l'appunto. Viceversa dovrà essere resettato nel caso in cui si desideri che sia proprio il coprocessore ad eseguire l'istruzione incriminata: ciò significa che anche se il coprocessore matematico è presente si può lavorare in emulazione, il tutto per testare la compatibilità di programmi su sistemi non dotati viceversa di coprocessore.

— MP («Monitor coProcessor») viene usato in unione al già citato bit TS indicante un «task switch», per poter effettuare l'operazione attraverso una «trap», piuttosto che generare un'eccezione. Detto così non promette nulla di buono, ma vale quanto detto in precedenza e cioè che a tempo debito verrà svelato l'arcano.

— PE («Protection Enable»): ecco qui una novità rispetto al 286. Quando questo bit è settato, viene abilitato il «Protected Mode», mentre quando viene resettato, il processore ritorna nel «Real Mode»: a differenza del 286, in cui tale

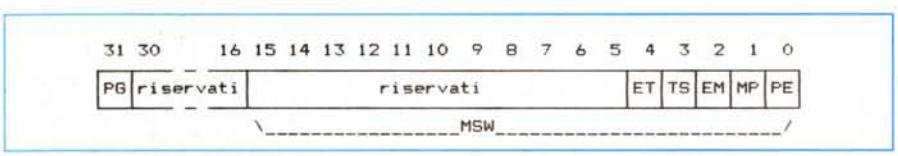

Figura 1 - Struttura interna del registro di controllo CR0, che nella sua parte meno significativa è praticamente coincidente con il registro MSW del 286: solo nella parte più significativa (disperso tra i bit «Intel Reserved») troviamo un bit nuovo.

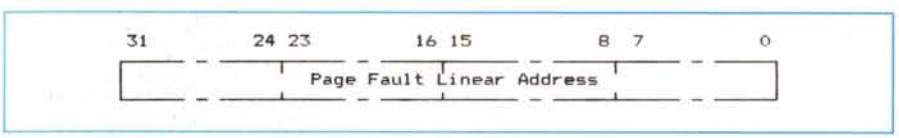

Figura 2 - Struttura del registro CR2, che contiene un indirizzo fisico a 32 bit, relativo all'ultima locazione di memoria che ha generato un'eccezione di «page fault».

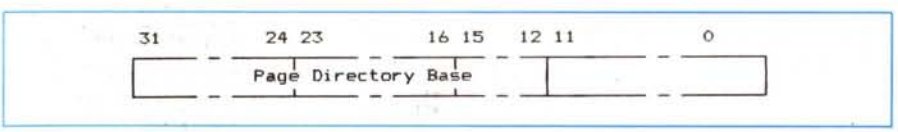

Figura 3 - Il registro CR3 contiene un indirizzo fisico a 20 bit della tabella detta «Page Directory Table», che gestisce la memoria di tipo «cache».

bit può essere solo settato con un'istruzione LMSW («Load Machine Status Word»), nel 386 tale bit può essere resettato (caricando il registro CR0 con un'istruzione di MOV apposita), per ritornare al modo «reale», ma la cosa non sarà così indolore come sembra, in quanto dovranno essere effettuate parecchie operazioni aggiuntive, di cui parleremo, eccetera eccetera..., ma che tutto sommato sono necessarie per far ritrovare al 386 un mondo in cui non esistono privilegi, protezioni e dove i registri devono avere valori opportuni.

Il registro CR2, che vediamo in figura 2, invece contiene un valore completo a 32 bit che rappresenta l'indirizzo «lineare» (appunto a 32 bit) della locazione di memoria che ha causato l'ultima condizione di «page fault», informazione aggiuntiva rispetto al valore dell'«error code» che viene posto, come è noto, nello stack del gestore di errori.

Il registro CR3, infine, che vediamo in figura 3, contiene nei 20 bit più significativi (i rimanenti 12 bit sono i soliti «Intel Reserved») il valore dell'indirizzo base fisico della cosiddetta «Page Directory Table», che gestisce la «cache memory» utilizzata dal 386 per velocizzare le sue funzioni.

Altri nuovi registri: i «Debug and Test Registers»

Ecco un'altra ottima novità del 386 e già sfruttabile grazie a quel meraviglioso programma che è il «Turbo Debugger» della Borland, nella versione specifica per il 386: si tratta di alcuni registri utilissimi nel debugging, in quanto consentono di implementare dei breakpoint di tipo «hardware».

Notevolissima è la differenza tra una gestione «software» ed una gestione «hardware» dei breakpoint: in quella software ci deve essere un programma, il debugger, che istruzione dopo istruzione deve verificare se l'indirizzo dell'istruzione che deve essere eseguita è proprio uguale a quello posto nella tabella di breakpoint.

Ciò comporta un notevole dispendio di tempo e non consente raffinatezze viceversa possibili con i breakpoint «hardware».

In questo caso infatti è il microprocessore ad avere una tabella interna di breakpoint (al massimo 4) ed è sempre lui a testare (all'interno dei chip e non da programma!!!) se l'indirizzo corrente coincide con uno dei quattro break predisposti: ma questo è niente.

Infatti con questo potente mezzo, si può porre un breakpoint anche sull'indirizzo di una locazione di memoria di dati (e non di programma, dunque) per far sì che scatti il break allorché, nel corso di un programma, si faccia riferimento ad una certa locazione di memoria: già questa innovazione (e ricordiamo che il tutto avviene all'interno del chip) fa comprendere subito di quanto è più potente il 386 rispetto non solo al 286, ma al 286 dotato di programmi di debugging sofisticati.

Rimandando a momenti migliori, riportiamo in figura 4 la struttura cumulativa degli 8 «Debug Registers» (da DR0 a DR7), dei quali due sono «Intel Reserved», e gli ultimi due dei quali servono, il primo, a «sorvegliare» lo stato dei breakpoint, mentre il secondo per stabilire le modalità di attivazione dei breakpoint stessi, secondo meccanismi parecchio complicati.

I «Test Registers» invece sono dei registri strettamente connessi con le tematiche relative al self-test del 386: infatti il nostro microprocessore, all'atto del reset e se il pin «Busy» è tenuto basso, può eseguire un self-test e cioè una serie di operazioni interne per saggiare la bontà dei circuiti e della logica interna, completamente trasparenti (e cioè invisibili) dall'esterno, se non per il fatto che il risultato del test è fornito come contenuto del registro EAX: se questo è nullo allora tutto va bene, altrimenti si potrebbero prendere provvedimenti in merito. Sorvoliamo per ora sull'utilizzazione dei due registri di Test, ma sottolineiamo il fatto che stranamente la stessa Intel ci avverte che i meccanismi collegati a tali due registri sono particolari per l'80386 e non è detto che possano essere mantenuti per i micro successivi (non abbiamo ancora avuto modo di verificare se nel 486 sono stati mantenuti, ma vi terremo informati..., n.d.r.): alquanto strano, come pure è strano che si chiamino TR6 e TR7; l'unica risposta potrebbe essere che i due registri sono l'unica «propaggine visibile» di un meccanismo interno che nel futuro scomparirà (all'utente, pur rimanendo fisicamente nel chip).

I modi di indirizzamento: altre novità

Ecco un altro campo in cui i progettisti dell'Intel hanno apportato alcune modifiche, aggiungendo, dal punto di vista del software, una nuova serie di possibilità di indirizzamento più generalizzato che non nei precedenti microprocessori.

Innanzitutto bisogna ricordare che il 386 nasce con 32 bit, ma può viceversa lavorare sia su singoli bit, sia su quantità da 8, 16 e 32 bit (registri e/o memoria a seconda delle scelte del programmatore): in totale ci sono 11 tipi di possibili modi di indirizzamento.

Andiamoli a vedere...

I primi due sono quelli più semplici e naturali e si riferiscono ad operandi di tipo registro e di tipo immediato: è evidente a cosa ci riferiamo, ma non fa mai male tornare un po' alle origini.

Nel primo tipo di indirizzamento l'operando risiede in un registro, che come sappiamo può essere a 8 bit (ad esempio CL), a 16 (ad esempio DX) oppure a 32 bit (ad esempio ESI): in questo caso il valore su cui opera l'istruzione si trova all'interno del registro specificato.

Nel secondo tipo di indirizzamento invece l'operando immediato risiede proprio nell'istruzione stessa e non potrà mai essere cambiato.

Tutte queste erano cose ben risapute: ma ora vediamo gli altri modi.

In particolare questi nove modi nascono dalla combinazione di quattro tipi di componenti base e che servono ad identificare l'indirizzo di memoria di un operando: ovviamente ed indipendentemente dal tipo di indirizzamento, valgono le stesse regole che già ben conosciamo e che sono un patrimonio in dotazione dei micro dell'Intel.

Ci stiamo riferendo ai meccanismi di calcolo dell'indirizzo fisico legati ad un «segment» e ad un «offset», che vengono gestiti in modo differente a seconda se ci si trova in «Real Mode» oppure in «Protected Mode»: in poche parole (anche per non ripetere alla noia concetti che anche i classici sassi conoscono) l'indirizzo fisico di una locazione di memoria viene calcolato sulla base di due quantità, che sono appunto il «segment» in cui si trova la locazione ed il ben noto EA («Effective Address»), sul calcolo del quale appunto si innestano nuovi procedimenti di calcolo.

L'Effective Address viene calcolato

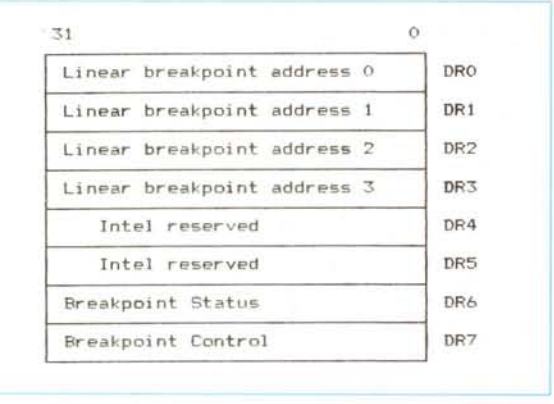

Figura 4 - L'insieme degli otto registri di Debug del 386, che già da soli sono un «fiore all'occhiello» del microprocessore.

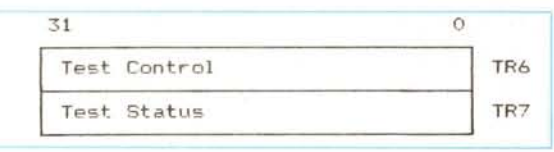

Figura 5 - I due «Test Register» (TR6 e TR7) vengono utilizzati per questioni riguardanti il self-test del 386 all'atto del reset.

```
 1  0000                           .MODEL TPASCAL
 2                                 .386
 3  0000                           .DATA
 4                                       ORG 0ABCDH
 5  ABCD  ????????                 ALFA    DD ?
 6  ABD1                           .CODE
 7  0000  66! FF 06 ABCDr          START:  INC ALFA
 8  0005  67! FF 03                        INC WORD PTR [EBX]
 9  0008  67! FF 83 00004FF2               INC WORD PTR [EBX + 4FF2H]
10  000F  66! 67! 03 92         +          ADD EDX,ALFA[EDX]
11        0000ABCDr
12  0017  66! 67! 8B 0C 5D      +          MOV ECX,[EBX * 2]
13        00000000
14  0020  66! 67! 8B 1C 1B                 MOV EBX,[EBX][EBX]
15  0025  66! 67! 8B 1C 5B                 MOV EBX,[EBX * 2][EBX]
16  002A  67! C7 04 83 0001                MOV WORD PTR [EBX][EAX * 4],1
17  0030  67! C7 04 FE 0002                MOV WORD PTR [EDI * 8][ESI],2
18  0036  66! 67! FF 8C 99      +          DEC ALFA[EBX * 4][ECX + 1000H]
19        0000BBCDr
20
21                                       END START
```

Figura 6 - Listato di un programmino di prova in TASM (Turbo Assembler) per vedere come vengono codificate le istruzioni che utilizzano i nuovi modi di indirizzamento del 386.

```
Turbo Debugger Log
CPU 80286
  cs:0000 66FF06CDAB      inc     dword ptr [ABCD]
  cs:0005 67FF03          inc     word ptr [ebx]
  cs:0008 67FF83F24F0000 inc      word ptr [ebx+00004FF2]
  cs:000F 66670392CDAB00+add      edx,[edx-000054331
  cs:0017 66678B0C5D0000+mov      ecx,[2*ebx]
  cs:0020 66678B1C1B      mov     ebx,[ebx+ebx]
  cs:0025 66678B1C5B      mov     ebx,[ebx+2*ebx]
  cs:002A 67C704830100    mov     word ptr [ebx+4*eax],0001
  cs:0030 67C704FE0200    mov     word ptr [esi+8*edi],0002
  cs:0036 6667FF8C99CDBB+dec      dword ptr [ecx+4*ebx-00004433
```

Figura 7 - Questo è invece il disassemblato dello stesso programmino di figura 5, ottenuto con il TD (Turbo Debugger), per vedere viceversa come vengono interpretate le nuove istruzioni del 386.

con la combinazione di quattro elementi (alcuni dei quali già noti) detti:
— «displacement»
— «base»
— «index»
— «scale»
sulla cui identificazione manterremo rigorosamente la nomenclatura originale inglese per non essere tentati di parlare di «spiazzamento» (orrore..., n.d.r.) come inutile italianizzazione del termine viceversa universale «offset».

Con «displacement» si intende un valore immediato (ad 8, 16 o 32 bit) posto all'interno dell'istruzione stessa e che indica l'indirizzo della locazione di memoria: infatti quando scriviamo:

DEC ALFA

intendiamo correttamente di voler decrementare il contenuto della locazione di memoria il cui indirizzo è dato dal valore simbolico ALFA (che deve essere posto in un data segment e seguito dalla direttiva DB, DW o DD, come ben sappiamo).

Perciò supponendo che ALFA faccia riferimento alla locazione posta all'offset 2ABDH (all'interno del data segment, ovvio!), ecco che nell'istruzione comparirà proprio quel valore 2ABDH, immutabile: noi che lavoriamo con un Assembler sappiamo che nell'istruzione c'è scritto l'offset di ALFA e non ci preoccupiamo certo di cosa ci porrà l'assemblatore stesso.

Viceversa debuggando il programma (in modo «non simbolico») troveremo una decodifica del tipo:

DEC word ptr [2ABDH]

che ci fa vedere appunto un «displacement».

L'elemento «base» è una generalizzazione del concetto già ben noto per averlo usato milioni di volte con l'8086 o l'80286: mentre però in questi due microprocessori come «base» può essere usato il registro BX (per dati nel data segment) oppure il registro BP (per dati nello stack segment).

Ovviamente questo fatto è molto riduttivo ed infatti nel 386 può essere usato come registro «base» un qualunque registro di quelli a 32 bit visti la scorsa puntata: mentre prima potevamo solo scrivere ad esempio MOV BYTE PTR [BX],7, ora abbiamo la possibilità di indirizzare la cella con il contenuto ad esempio di EAX, di EBX, ecc, scrivendo istruzioni del tipo:

MOV BYTE PTR ALFA[EDX],3

come pure del tipo:

INC WORD PTR ALFA[ESI]

L'altro elemento, l'«index», è ancora una volta l'estensione di quanto già conoscevamo per l'8086 e l'80286: finora potevamo usare appena DI ed SI, mentre ora possiamo usare uno qualunque dei registri a 32 bit (con l'eccezione di ESP).

Ecco che dunque possiamo scrivere istruzioni del tipo:

MOV BYTE PTR ALFA[EDX][EAX],3

come pure del tipo:

INC WORD PTR ALFA[ESI][EDI]

che sono un'estensione delle due precedenti e dove vediamo già cose strane, tipo l'uso contemporaneo di ESI e EDI, ora perfettamente legale.

Come pure è perfettamente legale un'istruzione del genere:

MOV EBX,[EBX][EBX]

la quale carica in EBX il contenuto (a 32 bit) della locazione di memoria posto all'offset calcolato dal contenuto di EBX sommato al contenuto... di EBX stesso.

In questa istruzione si vede in particolare quanta poca differenza c'è tra l'elemento «base» e l'elemento «index»: data dunque la generalità d'uso dei registri general purpose, in genere è inessenziale sapere quale dei due registri è «base» e quale è «index». Anche in istruzioni del tipo:

INC ALFA[EBP][EAX]

solo per dovere di cronaca EBP è «base» ed EAX l'«index», ma può essere anche considerato vero il viceversa...

L'ultimo elemento, lo «scale», è invece completamente nuovo ed è in un certo senso una sorpresa: si tratta di un fattore moltiplicativo (pari ad 1, 2, 4 oppure 8) che si può agganciare ad un qualunque registro di quelli visti in precedenza (eccetto ESP e per una sola volta nell'istruzione): si possono dunque avere istruzioni del tipo seguente:

MOV AX, [ESI*4]
MOV BX, [EBX][EBX*2]

come pure (e qui saltano fuori i 9 tipi nuovi di indirizzamento) tutte le possibili combinazioni dei quattro elementi, che abbiamo racchiuso nel piccolo programma di prova riportato in figura 6, anche per vedere come vengono codificate certe istruzioni.

Per inciso nella figura citata appare il file «.LST» prodotto dall'assemblatore TASM (Turbo Assembler) della Borland, mentre nella figura 7 riportiamo quello che viene disassemblato dal TD (Turbo Debugger) e che solo in prima analisi è leggermente differente da quello che viceversa abbiamo impostato noi.

Con questo terminiamo la presente puntata e diamo l'appuntamento alla prossima dove proseguiremo il nostro studio sul 386.