ASSEMBLER 8086 8088

di Pierluigi Panunzi

seconda parte

Il set di istruzioni
Istruzioni di gestione interrupt e coprocessori

Come già anticipato nella scorsa puntata, in questa parleremo delle ultime istruzioni che ci sono rimaste da analizzare, relative ad argomenti un po' più complessi di quelli trattati finora, in quanto in generale legati più a considerazioni hardware che a concetti di software. Per comprendere meglio il funzionamento e l'utilità delle istruzioni di cui sopra, premetteremo una parte teorica riguardante la gestione degli interrupt da parte dei due microprocessori sin qui analizzati, l'8086 e l'8088.

Terminata poi l'analisi di queste ultime istruzioni, lo promettiamo, effettueremo «il grande salto» passando dai piccoli '86 ed '88 (i «motori» dei personal computer PC, XT dell'IBM e della maggior parte dei «compatibili») al ben più potente 80286, ormai da parecchio tempo utilizzato (o meglio sottoutilizzato) dei modelli «AT» della «casa madre», in un gran numero di compatibili, nonché da ultimo nei «primi modelli» della nuovissima serie «Personal System 2» sempre dell'IBM.

Solo dopo aver analizzato il '286 potremo dunque affrontare anche il discorso sull'80386, anch'esso già utilizzato in personal computer (Compaq 386, ad esempio) e dai modelli di punta della già citata serie «PS/2».

Non è evidentemente possibile parlarne già da adesso, scavalcando a pie' pari il '286, in quanto il «mostro a 32 bit» è particolarmente difficile da capire, specie se non si conoscono le caratteristiche intrinseche del '286, che già di per sé sono alquanto complicate.

Comunque ritorniamo al nostro «semplice» microprocessore, senza precorrere di troppo i tempi.

La gestione degli interrupt

Nell'8086/88 si possono avere due tipi di interrupt, ai quali il microprocessore risponde esattamente allo stesso modo: si tratta degli «interrupt software» e degli «interrupt hardware».

Mentre questi ultimi dovrebbero essere ben noti in quanto si tratta dei «soliti» interrupt provenienti da dispositivi (di interfaccia) esterni e presenti in un qualsiasi mondo gestito da un qualunque microprocessore, gli «interrupt software» sono, come dice il nome, generati, «scatenati» da programma, per mezzo di particolari istruzioni, che però solo in prima analisi possono servire per «simulare» l'arrivo di un interrupt e perciò testare il comportamento del programma in tali condizioni.

Iniziamo dunque dagli interrupt hardware, in quanto più semplici da comprendere.

In particolare un interrupt arriva al microprocessore attraverso il pin «INTR» (che sta per «INTerrupt Request»), provenendo in genere da un dispositivo esterno (USART, Programmable Interrupt Controller, Timer, ecc.).

In generale il dispositivo esterno può generare una richiesta di interrupt per il sopraggiungere di un certo evento tra i tanti che possono provocare la richiesta di interruzione: ad esempio se il dispositivo interfacciante è un USART (Universal Synchronous Asynchronous Receiver Transmitter), questo in generale potrà effettuare richieste di interrupt:

— quando ha ricevuto un byte dalla linea sincrona o asincrona;

— quando ha appena trasmesso l'ultimo byte sulla linea ed ha bisogno di un nuovo byte da parte della CPU;

— quando ha riscontrato un errore durante la ricezione di un messaggio (errore di parità, di perdita di sincronismo, ecc.);

— quando si accorge dell'impossibilità di trasmettere un byte sempre per problemi di linea.

In tutti questi casi dunque (ed il nostro è un esempio puramente teorico) il dispositivo emette un interrupt, portando allo stato logico «1» la linea collegata al pin «INTR» del microprocessore. Come farà quest'ultimo a capire quale è stata la causa che ha generato l'interrupt (nel nostro caso le possibilità erano 4)? Ci sono due metodi di risposta a questo interrogativo.

Se il dispositivo è della stessa casa costruttrice dell'8086/88 e cioè è dell'Intel, allora in generale sarà proprio il dispositivo ad informare il microprocessore secondo un meccanismo di interscambio molto semplice e che ora analizziamo per grandi linee:

— il disposivo lancia dunque un interrupt al micro, in conseguenza del verificarsi di un certo evento, l'«N»-esimo, tra un

certo insieme di eventi possibili;

— il micro «sente» che la linea «INTR» è a livello logico «1», termina l'istruzione che stava eseguendo in quel momento ed invia, attraverso il pin «INTA» («INTerrupt Acknowledge»), il cosiddetto «acknowledge» (consenso, risposta) al dispositivo, come per dire:

«Guarda, che io ho ricevuto il tuo interrupt, ora aspetto ulteriori informazioni...»;

— a questo punto il dispositivo emette sul Data Bus (ad 8 bit) un valore esadecimale corrispondente all'evento verificatosi: nel nostro caso semplificato porrà sul bus il valore «N»;

— il microprocessore leggerà questo valore («N»), lo moltiplicherà per 4 ed otterrà così l'indirizzo di una locazione di memoria dove troverà a sua volta l'indirizzo completo (formato da offset e da segment) della routine che deve essere eseguita per «onorare» l'interrupt: il controllo passerà dunque a tale routine, al termine della quale il controllo ritornerà di nuovo alla routine interrotta;

— nel frattempo il dispositivo, ricevuto l'acknowledge, provvederà a riportare a «0» lo stato della linea di «Interrupt Request» per evitare che il microprocessore riesegua un'altra volta la routine di interrupt alla fine di quella che sta eseguendo, per evitare cioè che il micro entri in un ciclo infinito.

Con il che l'interscambio è finito.

Nel caso invece che il dispositivo non sia dell'Intel, allora in generale sarà presente al suo interno un registro che conterrà il valore «N» relativo all'interrupt generato: sarà allora il micro a leggere tale registro e a prendere i provvedimenti del caso.

Comunque quest'ultima non è che una semplificazione, in quanto in generale ci saranno più dispositivi «generatori di interrupt» ed il povero micro non sarebbe in grado di sapere chi è stato a generare la richiesta: ci vuole perciò un dispositivo «selettore» che tenga conto pure delle dovute «priorità» e cioè del fatto che in caso di contemporaneità di richieste di interrupt, uno dei due interrupt deve essere eseguito per forza di cose prima dell'altro.

A tale scopo dunque (e ciò si ha ad esempio in tutti i personal citati in precedenza) esiste un componente chiamato «Programmable Interrupt Controller» (PIC, l'8259 dell'Intel) che può ricevere fino ad otto interrupt da altrettanti dispositivi esterni ed è proprio lui ad indicare al micro, nella fase di acknowledge, il numero del dispositivo «incriminato»: l'8086/88 in base a questo valore (moltiplicato per 4) otterrà un indirizzo, andrà a leggere nella memoria l'indirizzo completo della routine di interrupt e a lei passerà il controllo.

Sarà questa routine che ora andrà a leggere il registro interno del dispositivo che ha lanciato l'interrupt, per vedere a sua volta quale è stata la causa che ha provocato l'emissione della richiesta di interruzione.

Ovviamente questa è un'analisi parecchio semplificata di quello che succede in questi frangenti: non ce ne vogliano i lettori più esperti in quanto crediamo che ciò sia risultato viceversa comprensibile a chi conosce poco la materia.

Vediamo dunque meglio il discorso della «moltiplicazione per 4» ed i relativi passaggi con letture in memoria.

L'«Interrupt Vector Table»

Abbiamo dunque visto che a partire da un'unica linea di richiesta di interrupt, il microprocessore 8086/88 ha la possibilità di decidere una di 256 possibili routine di gestione dell'interrupt in base al valore fornito sul Data Bus subito dopo la fase di acknowledge dal dispositivo che ha generato l'interruzione: ciò come detto vale solo per dispositivi dell'Intel, mentre per altri componenti è richiesta la presenza di un Interrupt Controller.

Ora queste 256 routine possono essere poste in memoria in un qualsiasi punto, a partire dall'indirizzo fisico (a 20 bit) 00400H per arrivare all'indirizzo 0FFFF0H: l'indirizzo della singola routine sarà dunque formato dalla coppia offset-segment (2 word pari a 4 byte) e perciò l'insieme degli indirizzi delle 256 routine (pari ad 1 kbyte), detto «interrupt vector table» è posto a partire dall'indirizzo fisico 00000H, fino all'indirizzo 003FFH compreso.

Dal momento che ad ognuna delle 256 routine corrispondono 4 byte ecco il motivo della moltiplicazione per 4 (shift a sinistra di due bit) del «numero» della routine per poterne ricavare l'indirizzo di partenza («entry point») sotto forma di coppia offset-segment.

Ecco che ad esempio, se la routine di risposta dell'interrupt numero 23H è posta in memoria all'indirizzo 43DFH:0000H (pari all'indirizzo fisico 43DF0H), allora tale indirizzo dovrà essere posto in memoria (dal sistema operativo) all'indirizzo dato da 23H moltiplicato per 4 e cioè all'indirizzo 0008CH ed in particolare nella word posta a 0000H:008CH (abbiamo espresso l'indirizzo fisico nel modo più usuale) ci sarà il valore 0000H (l'offset dell'entry point della routine di gestione dell'interrupt), mentre nella word successiva, posta all'indirizzo 0000H:008EH, ci sarà il valore 43DFH, pari appunto al segmento della routine di gestione dell'interrupt.

Ovviamente il riempimento della tabella dei vettori di interrupt deve essere fatto a cura del sistema operativo «prima» che un qualsiasi interrupt raggiunga il microprocessore, altrimenti si avrebbero le ovvie conseguenze di un malfunzionamento se non di un completo crash del sistema: è buona norma poi prevedere «tutti» gli interrupt e perciò prevedere tutti gli elementi della tabella.

Dal punto di vista programmativo, all'atto dell'esecuzione della routine di interrupt, il microprocessore provvede automaticamente a salvare nello stack il contenuto dei flag nonché l'offset ed il segment dell'istruzione successiva a quella che stava eseguendo del programma che è stato dunque interrotto: al termine della routine di interrupt, l'apposita istruzione IRET (della quale parleremo dopo) provvederà a ripristinare il CS, l'IP ed i flag.

Il «Non Maskable Interrupt» (NMI)

Abbiamo parlato, sempre nell'ambito degli interrupt hardware, di interrupt mascherabili, intendendo con tale termine la possibilità consueta da parte del microprocessore di poter ignorare un interrupt, semplicemente grazie ad un'istruzione e perciò via software (come vedremo in seguito).

Esiste invece, come per tutti i micro che si rispettino, un interrupt cosiddetto non-mascherabile, che il micro non può in alcun modo ignorare, qualsiasi cosa stia facendo nel momento di arrivo: si tratta in generale di interrupt legato ad un evento esterno la cui gravità compromette il corretto funzionamento del sistema.

Mentre in generale, in sistemi ad alto rischio, l'NMI è connesso ad un circuito che controlla istante per istante la tensione di alimentazione e provvede ad attivare un circuito di protezione della memoria RAM in caso di abbassamento repentino dell'alimentazione, nel caso dei personal computer «veri» o «compatibili» tale interrupt è strettamente legato alla memoria dinamica ed in particolare al controllo del bit di parità: quando in un qualsiasi momento il circuito di controllo della parità della RAM trova un errore (indicante un malfunzionamento di un chip di memoria), viene generato l'NMI, il cui effetto è di blocare il funzionamento del sistema con la visualizzazione sul display del messaggio «Parity Error».

Il tutto è alquanto drastico ed al povero programmatore che si è trovato in tale sfortunata circostanza non resta che sperare che il malfunzionamento sia temporaneo e tentare di ricominciare tutto daccapo, dal momento che comunque si perde il controllo del sistema ed in particolare di tutto quanto risiedeva in memoria (il programma!).

Comunque se proprio si vuole evitare questo disastro, magari sperando che il difetto appaia in un chip relativo ad una zona di memoria «lontana» dal punto in cui stiamo lavorando oppure sperando che sia proprio il chip di parità ad essersi rotto (ma questi eventi così favorevoli al pro-

grammatore, per la ben nota «Legge di Murphy» non accadono MAI!), ecco che i saggi progettisti dell'IBM hanno previsto che anche l'interrupt non mascherabile per eccellenza sia mascherabile via hardware, per mezzo di una porta, però comandabile via software.

Per quanto riguarda il punto di vista software della faccenda, all'NMI corrisponde l'interrupt numero 2 e perciò l'indirizzo della routine di gestione dell'NMI è posto nell'Interrupt Vector Table all'indirizzo 0000H:0008H.

Lo smaliziato programmatore potrebbe far sì che un dispositivo esterno risponda proprio con il valore 2 dopo la fase di acknowledge del micro ad un interrupt appena lanciato: per il software del micro tutto va come se fosse arrivato un NMI, ma viceversa mascherando gli interrupt via software, allora tale interrupt non arriverebbe più, a differenza dell'NMI, ovviamente.

Il «Reset»

Visto che ci siamo, parliamo anche di questo evento esterno, che giunge al piedino «RESET» del nostro 8086/88: è ovviamente quanto di più non-mascherabile (almeno per il software, in quanto via hardware si può fare di tutto!) esista per il micro. Ovviamente la fase di «reset» si ha all'inizio dei tempi, all'accensione della macchina oppure alla pressione di un apposito pulsantino e dal punto di vista softwaristico prevede l'azzeramento dell'Instruction Pointer (IP) ed il settaggio del registro CS (Code Segment) al valore 0FFFFH: ciò equivale ad imporre un salto alla locazione di memoria di indirizzo 0FFFFH:0000H, pari ad un indirizzo fisico pari a 0FFFF0H.

A tale indirizzo presumibilmente si troverà un «long JMP» alla routine iniziale, dal momento che a partire dall'indirizzo fisico 0FFFF0H abbiamo appena 16 byte a disposizione (ricordiamo che il nostro micro può indirizzare 1 Mbyte di memoria).

I «Software Interrupt» L'istruzione INT

Tutto quanto abbiamo visto finora riguarda il software solamente per quanto riguarda indirizzi di routine e relativi salti: abbiamo però parlato dell'esistenza dei cosiddetti «Software Interrupt», generabili appunto via software, per mezzo dell'istruzione INT.

Già dalla sintassi dell'istruzione, che è

```
INT n
```

dove «n» è un valore intero compreso tra 0 e 255, si può arguire facilmente che l'esecuzione di un'istruzione «INT n» equivale all'esecuzione della routine di interrupt relativa ad un dispositivo esterno che ha fornito sul data bus il valore «n» dopo la fase di acknowledge dell'interrupt appena inviato.

Abbiamo detto che equivale «all'esecuzione» della routine di interrupt e perciò anche in questo caso viene salvato nello stack il contenuto dei flag e l'indirizzo di ritorno dell'istruzione successiva: a parte i flag il tutto equivale al funzionamento nel caso di un'istruzione di CALL (di tipo «long») ed è questo il motivo che tale istruzione, lungi dall'essere usata per simulare il funzionamento di una routine di interrupt, viene più proficuamente utilizzata al posto di una CALL, con il vantaggio della leggibilità e dell'occupazione di memoria (2 byte dell'istruzione INT contro i 5 della CALL «lunga»).

L'unico scotto che si paga è nella maggiore durata dell'istruzione in termini di cicli di clock, praticamente raddoppiato: 28 clock per una CALL «inter-segment» e 51 per una INT.

Il vantaggio dal punto di vista della leggibilità si ha nel fatto che ai vari INT «n» possono essere associate altrettante routine «importanti» e richiamabili semplicemente, tanto è vero che l'MS-DOS si basa su tale istruzione per l'attivazione di routine di sistema e per la gestione delle risorse hardware interne ai PC (video, tastiera, porte parallele e seriali, timer, ecc).

Un caso particolare dell'istruzione INT è quello in cui «n» vale 3 (istruzione «INT 3») che ha un codice operativo ad un byte solo (pari a 0CCH), e che corrisponde in genere alla gestione software di «break-point» in programmi di debugging: in tal caso basta sostituire l'op-code dell'istruzione posta nell'indirizzo di break-point con il valore 0CCH ed automaticamente il micro eseguirà la routine relativa (il cui indirizzo è posto nell'Interrupt Vector Table all'indirizzo 0000H:000CH).

Eseguita tale routine basta ripristinare tale byte con il vecchio valore ed il gioco è fatto.

Analizziamo ora in dettaglio il funzionamento interno del micro all'atto dell'esecuzione dell'istruzione INT «n»:

— viene salvato nello stack il contenuto del registro dei flag;

— vengono resettati i due flag IF e TF (rispettivamente «Interrupt Flag» e «Trap Flag»), dei quali parleremo dopo;

— viene salvato nello stack il CS;

— viene posto in CS il contenuto della locazione di memoria di indirizzo pari a «n * 4 + 2» dove «n» è il valore posto nel secondo byte dell'istruzione (o vale automaticamente 3 se l'istruzione è la «INT 3» con opcode singolo) e dove il «+2» si riferisce al fatto che nell'Interrupt Vector Table il segmento è posto nella word più significativa, come al solito;

— viene salvato nello stack il contenuto di IP;

— viene posto nel registro IP il contenuto della locazione di memoria di indirizzo pari a «n * 4».

Come si vede dunque i flag rimangono inalterati (anzi, salvati!) ad eccezione dei già citati IF e TF.

L'istruzione INTO

L'istruzione in esame è un altro caso particolare dell'istruzione INT, specializzata per il caso in cui risulti settato il flag di overflow (OF): infatti il nome INTO deriva appunto da «INTerrupt if Overflow».

In particolare, in caso di overflow, con questa istruzione viene automaticamente attivato l'interrupt 4 e tutto va come se al posto di INTO ci fosse la sequenza

```
        JNO SOTTO
        INT 4
SOTTO:  ...
```

in cui appunto nella INTO è inglobato anche il test sul bit OF. Per completezza ed in analogia con quanto fatto finora, analizziamo in dettaglio il funzionamento dell'istruzione INTO:

— innanzitutto viene testato lo stato del flag di overflow e nel caso in cui esso risulti resettato allora si passa alla prossima istruzione del programma;

— vengono salvati nello stack i flag;

— vengono resettati i due flag IF e TF;

— viene salvato nello stack il CS;

— viene posto in CS il contenuto della word posta all'indirizzo 0000H:0012H, corrispondente al valore ottenuto dato dalla formula solita (n * 4 + 2) dove «n» ora vale 4;

— viene salvato nello stack il contenuto di IP;

— viene posto nel registro IP il contenuto della locazione di memoria di indirizzo 0000H:0010H, pari al valore ottenibile dalla formula «n * 4», ponendo n = 4, come già detto.

L'istruzione IRET

L'istruzione IRET (che deriva da «Interrupt RETurn») è l'istruzione che solitamente chiude le routine di gestione degli interrupt ed in particolare serve a ripristinare lo stato del CS, dell'IP e dei flag, che come sappiamo era stato salvato nello stack all'atto del riconoscimento di un interrupt e subito prima dell'attivazione della routine stessa.

In dettaglio l'istruzione IRET effettua le seguenti operazioni:

— pone nel registro IP il valore salvato nello stack;

— ripristina il registro CS con il secondo valore che era stato posto nello stack;

— ripristina lo stato dei flag, allo stesso modo in cui opera l'istruzione POPF, della quale abbiamo parlato in una puntata precedente e cioè andando ad alterare i soli bit del registro di flag, corrispondenti effettivamente a flag: ricordiamo infatti che non ci sono 16 flag, ma appena 9 e perciò alcuni bit sono «don't care».

Le istruzioni CLI e STI

Le due istruzioni in esame sono utilissime in quanto consentono rispettivamente di disabilitare ed abilitare la ricezione di interrupt esterni, gli «hardware interrupt», per mezzo del «flag di interrupt» (IF), che funziona da flip-flop di abilitazione o meno degli interrupt.

In particolare CLI («CLear Interrupt flag») resetta l'IF e perciò disabilita la gestione degli interrupt, mentre STI («SeT In-

terrupt flag») pone ad «1» tale flag riabilitando di nuovo la possiblità del micro di gestire gli interrupt esterni.

L'istruzione HLT

Con questa istruzione, che deriva ovviamente il suo nome da «HaLT», forza il microprocessore ad entrare nello stato di «halt», in cui non fa nulla, se non aspettare supinamente l'arrivo di un interrupt esterno (se abilitati precedentemente) oppure addirittura di un reset.

Si tratta di un'istruzione che come detto non esegue alcuna operazione (ma dura 2 cicli di clock ogni volta ed è perciò la più breve in assoluto) e deve essere usata solo in rari casi disperati in cui qualsiasi altra operazione diversa da un reset (o da un improbabile interrupt esterno di «salvataggio») sarebbe deleteria.

Sui PC o compatibili è usata solo all'interno della ROM contenente il BIOS ed in particolare viene usata durante il cosiddetto POST («Power On Self Test») e cioè durante il test che il computer fa inizialmente per verificare il corretto funzionamento dei suoi componenti principali: se ad esempio la CPU non funziona correttamente a livello registri interni, ecco che viene effettuato un salto ad un'istruzione HLT, così come accade nel caso in cui il sistema riscontri un errore di parità da parte della memoria RAM.

L'istruzione WAIT

È questa un'altra istruzione in particolar modo legata ad un evento esterno e serve per sincronizzare l'esecuzione delle istruzioni di un programma in base allo stato della linea connessa all'8086/88 per mezzo del pin «TEST».

In particolare quando viene incontrata l'istruzione di WAIT, il microprocessore entra in uno stato di «wait» (attesa), andando contemporaneamente a testare lo stato della linea «TEST»: se tale linea è in stato «attivo» e cioè «0» allora l'esecuzione del programma riprende con l'istruzione successiva, mentre viceversa un «1» su tale pin lascia il micro nello stato di «wait».

Altra possibilità per il micro di uscire, seppur temporaneamente, da tale stato, è l'arrivo di un interrupt esterno (sempre se abilitato).

Al termine della routine di gestione di tale interrupt, però, il controllo ripasserà ancora una volta all'istruzione di WAIT: è infatti un caso particolare in cui nello stack non viene salvato l'indirizzo dell'istruzione successiva, ma dell'istruzione stessa, caso più unico che raro tra le istruzioni del nostro microprocessore.

L'istruzione ESC

L'istruzione ESC («ESCape») è un'istruzione particolarissima, in quanto innesca un meccanismo tale che la successiva istruzione viene semplicemente «letta» dall'8086/88 e posta sul bus, per essere eseguita da un altro processore esterno all'8086/88, che in generale può essere il «coprocessore matematico» 8087, sempre dell'Intel.

Tale istruzione dunque inibisce la possibilità del microprocessore di eseguire l'istruzione successiva, la cui lunghezza in byte è gestita dal coprocessore: quest'ultimo cioè leggerà (per mezzo dell'8086/88) tanti byte quanti sono previsti per l'istruzione il cui opcode è proprio il byte successivo al byte «ESC», eseguirà l'operazione prevista dall'istruzione letta e poi ricederà il controllo all'«host processor».

Su quest'ultimo fatto ci sono da aggiungere alcune considerazioni: di solito il coprocessore matematico viene connesso all'«host» per mezzo di un certo numero di linee. In particolare l'uscita «BUSY» dell'8087 viene connessa all'ingresso «TEST» dell'8086/88 in modo tale da consentire la sincronizzazione tra i due processori.

Supponendo dunque tale configurazione e connessione hardware, vediamo che nel caso che in un programma venga incontrata una sequenza del genere:

```
...
MOV ALFA,AX
ESC istruzione dell'8087
MOV CX,SI
...
```

L'8086/88 eseguirà l'istruzione MOV ALFA,AX e poi incontrerà l'ESC con il che cederà il controllo all'8087 per leggere l'«istruzione dell'8087».

A questo punto il coprocessore rilascerà il controllo al «master» e si farà intanto i suoi calcoli (che magari richiedono parecchi cicli di clock), ma l'importante è che subito riprenda l'esecuzione anche l'8086/88, con il che si potrebbe perdere la sincronizzazione in casi non del tutto particolari: infatti supponiamo che l'«istruzione dell'8087» sia un'operazione che faccia riferimento ad una locazione di memoria, che verrà dunque alterata alla fine dell'istruzione stessa.

Ora se una delle istruzioni successive del programma fa riferimento a tale locazione di memoria, può accadere che il micro vi faccia riferimento prima ancora che in tale locazione sia posto un valore aggiornato.

In questo caso dunque il fatto di avere una esecuzione «parallela» di due processori (un vero e proprio «multi-processing») è senza dubbio deleterio.

Per ovviare a tale inconveniente basta far precedere il «prefisso» WAIT all'istruzione di ESC che fa riferimento alla locazione di memoria incriminata, in modo tale che il controllo venga ridato all'«host» solo al termine dell'istruzione del coprocessore, perdendo in questo caso il «parallelismo» di operazioni, che però può essere ampiamente recuperato con altre istruzioni che non fanno riferimento alla memoria.

L'istruzione LOCK

Siamo arrivati dunque all'ultima istruzione: si tratta di un'altra istruzione particolarissima ed utilizzabile in sistemi multiprocessor (ad esempio sistemi dotati di due processori veri e propri, al limite due 8086/88), per inibire l'accesso a risorse comuni all'altro processore: tali risorse comuni in genere sono alcune particolari zone di memoria «condivise» dai due processori ed alle quali non possono e non devono accedere contemporaneamente tutti e due i microprocessori.

Tale particolare gestione di risorse aggiuntive richiede una particolare attenzione nella progettazione dell'hardware, per mezzo del quale sia possibile proibire l'accesso al bus comune agli altri processori, per tutto il tempo che l'istruzione successiva alla LOCK lo richieda.

Anche in questo caso, come quello visto in precedenza, il «parallelismo» viene interrotto d'ufficio, per evitare ulteriori sgradevoli conseguenze.

Come si può vedere nelle due situazioni indicate, il comportamento è praticamente lo stesso e si potrebbe pensare ad un'inutile duplicazione: invece la spiegazione è molto semplice. Nel caso del coprocessore matematico, si tratta di un componente praticamente «sovrapposto» all'8086/88, con il quale condivide in generale tutti i segnali di controllo, oltreché ovviamente il Data Bus e l'Address Bus.

Inoltre l'8087 effettua anche lui la decodifica delle istruzioni, man mano che vengono lette dall'«host», in parallelo, ma non trovando mai il prefisso «ESC» non ne eseguirà alcuna.

Viceversa nel caso di presenza dell'ESC, abbiamo visto cosa succede con problemi riguardanti la sincronizzazione e perciò l'accesso ad una risorsa comune (la locazione di memoria incriminata): l'opportuna presenza dell'istruzione di WAIT serve a risolvere tutti i problemi.

Nel caso invece di più processori («veri») nello stesso sistema, che perciò è un «multi-processor system», in generale ogni processore (uno dell'Intel e magari l'altro della Motorola, tanto per fare un esempio) possiederà una sua zona di memoria EPROM contenente il proprio programma, ma necessariamente (per prima cosa per problemi di costi) avranno in comune la memoria RAM.

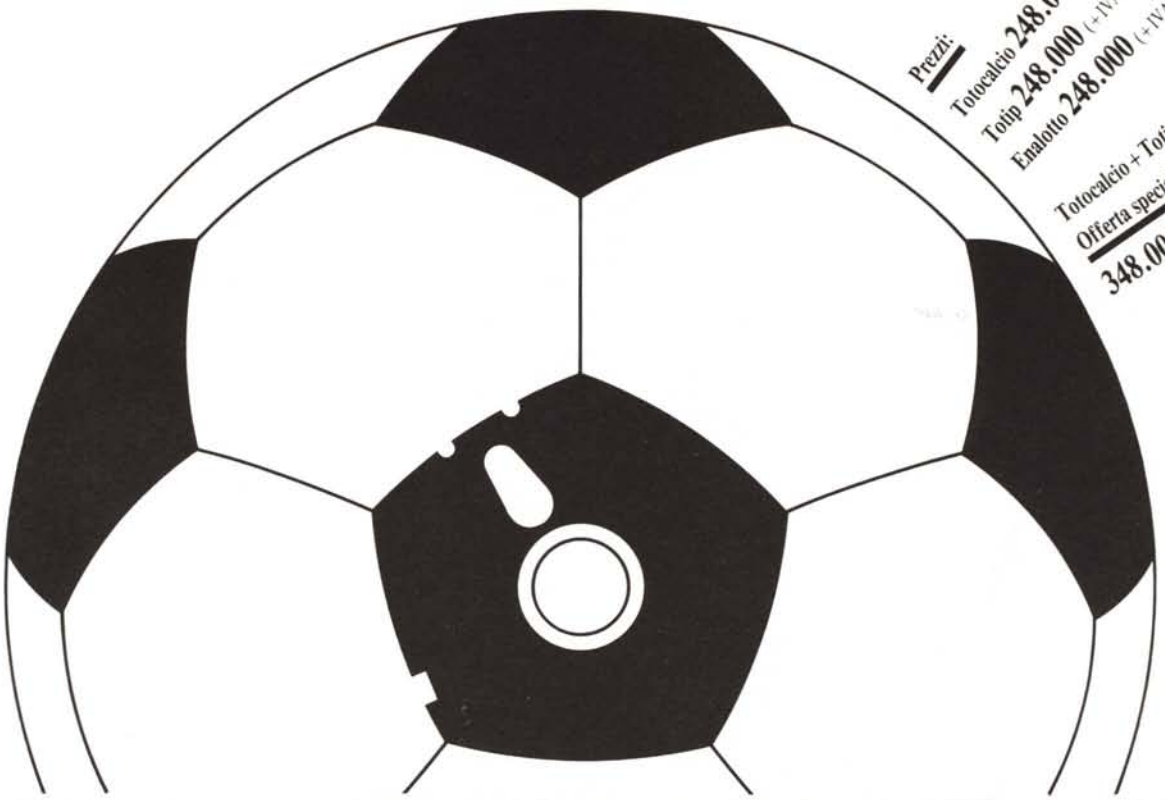
Ecco che dunque i due o più processori lavorano l'uno indipendentemente dall'altro, ignorandosi a vicenda se non laddove uno dei due od entrambi contemporaneamente entrano in una zona «comune» dove si creerebbero sicuramente grossi problemi di condivisione.

La soluzione in questo caso è necessariamente differente che nel caso del coprocessore matematico, dal momento che i programmi che ogni processore sta eseguendo sono completamente differenti (ed ovviamente scritti in «lingue» diverse!) e prevede dunque l'uso dell'istruzione LOCK per inibire l'accesso all'altro processore alla memoria condivisa.

Con questo concludiamo questa lunga e pesante puntata e diamo l'appuntamento per il prossimo numero allorché inizieremo a parlare dell'80286, in una rubrica che sicuramente avrà un altro titolo, ma scritta sempre dalla stesso redattore. MC

Regalati un 13!

è una proposta J.soft



Prezzi:
Totocalcio 248.000 (+IVA 9%)
Totip 248.000 (+IVA 9%)
Enalotto 248.000 (+IVA 9%)

Totocalcio + Totip + Enalotto:
Offerta speciale
348.000 (+IVA 9%)

13! Il programma per vincere al Totocalcio, al Totip, all'Enalotto. 13!

13! consente finalmente ad ogni giocatore di elaborare con tutta la potenza e la semplicità d'uso del **personal computer**, piccoli, grandi e "mega" sistemi.

Metodi di riduzione utilizzati

- Sistema a correzione multipla di errori: definizione e selezione dei risultati come Basi, Errori e Sorprese.
- Sistema derivato a condizione multiple: selezione delle colonne contenenti un numero minimo e massimo di segni 1-X-2 e con un numero minimo e massimo di 1-X-2 consecutivi.

Definizione del pronostico

- Impostazione di un nuovo pronostico o modifica di uno già definito.
- Definizioni delle squadre e dei segni componenti il pronostico.
- Utilizzo di un metodo di riduzione, di tutti o di nessuno.
- Elaborazione con estrazione delle colonne che hanno soddisfatto i vincoli impostati (con calcolo del rapporto di riduzione ottenuto e della percentuale delle colonne selezionate).
- Registrazione e/o stampa delle colonne ottenute e delle copie della definizione di sistema.
- Possibilità di modificare a piacere il pronostico (ridefinizione dei vincoli, dei metodi e dei segni in schedina).
- Possibilità di gestire più pronostici per settimana.
- Definizione del costo della singola colonna e calcolo del costo del sistema selezionato.

Spoglio

- Definizione della colonna vincente.
- Visualizzazione e stampa di tutte le colonne vincenti e calcolo della vincita totale.
- Gestione delle partite non valide.
- Se non è stata ottenuta una vincita di prima categoria, 13! presenta in dettaglio gli errori commessi rispetto alla colonna vincente.
- In caso di mancata vincita, 13! comunica il numero di colonne in cui si è ottenuto il miglior punteggio.
- Sistema operativo: MS-DOS 2.0 e successivi.

J.soft

Distributore per l'Italia

Viale Restelli, 5 - 20124 Milano
Tel. 02/6888228-683797-6880841/2/3