1 L'interrupt nell'8086 e nel PC

In questo capitolo si studia in dettaglio il funzionamento del sistema d'interruzione dell'8086, che è un significativo esempio di interrupt vettorizzato. Poi si spiega il più tipico controllore d'interruzione, cioè l'8259, infine si illustra l'architettura del sistema d'interruzione del PC.

1.1 Il sistema d'interruzione dell'8086

Come già accennato nel capitolo precedente la sequenza d'interruzione di una CPU 8086 è più complicata di quella descritta in quel capitolo.

Interrupt e flag

Abbiamo già anticipato che esiste un'istruzione specifica per il ritorno da una ISR e che il suo mnemonico è **IRET**.
La domanda che ci si dovrebbe porre ora è la seguente: "perché esiste la IRET, non basterebbe una RETF (RET far)?".
Dopotutto il ritorno dall'interruzione, così come descritto nel capitolo precedente, è uguale a quello di una procedura: la CPU trasferisce nel Program Counter ciò che "emerge" dallo stack.
La ragione dell'esistenza della IRET è che la sequenza di interruzione di un 8086 salva nello stack anche il valore attuale dei flag.
Vediamo di giustificare questa operazione che, se non fosse indispensabile, sarebbe piuttosto costosa, dato che viene fatta sempre dalla CPU, ad ogni e ciascuna sequenza di interrupt. Per aiutarci usiamo la Figura 1, che contiene una ISR di esempio.
In Figura 1 sono illustrate alcune righe del programma interrotto. Ricordando che una ISR può scattare in ogni istante, e non solo quando il programmatore lo vuole, la Figura 1 mostra, al punto <1>, un'occasione in cui l'arrivo di un interrupt sarebbe del tutto inopportuno e potrebbe causare danni, se la CPU non salvasse i flag.

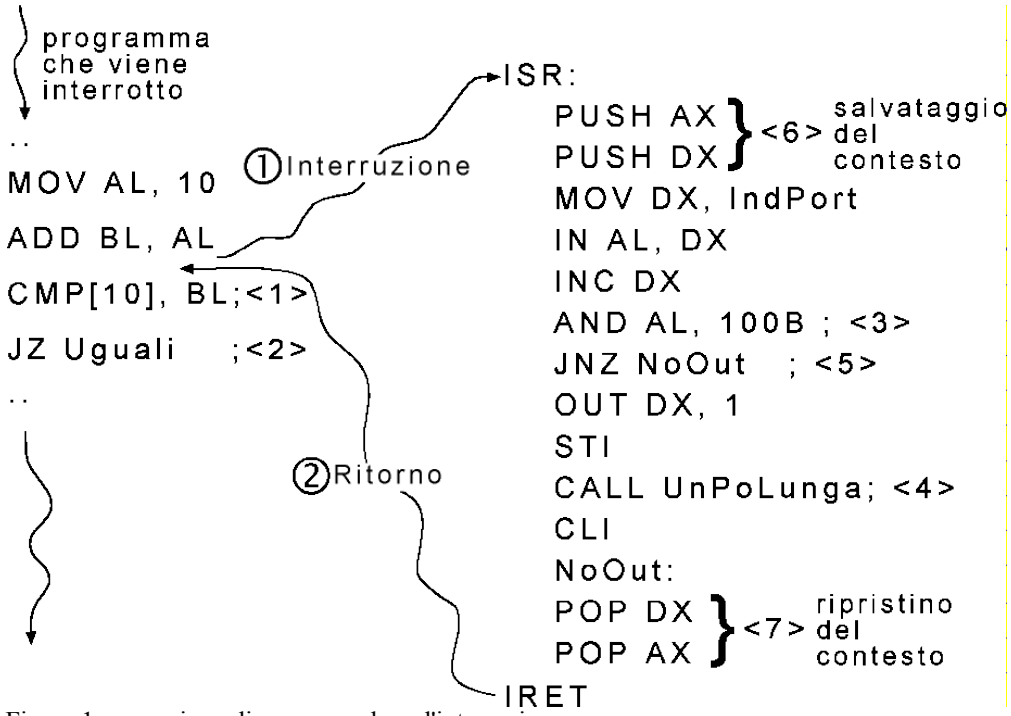

Figura 1: esecuzione di una procedura d'interruzione

Infatti l'istruzione <2> fa affidamento sui flag prodotti dall'istruzione <1>, la ISR però li modifica, l'ultima volta nell'istruzione <3>, o nella chiamata <4>.
Dunque il programma scritto in Figura 1 funziona regolarmente, ma solo perché la CPU salva automaticamente nello stack tutti i flag prima di saltare alla ISR, e perché l'istruzione IRET non ripristina solo l'indirizzo di ritorno ma anche tutti i flag generati dalla CMP dell'istruzione <1>.
Nelle più importanti CPU a 16 e 32 bit, e non solo nell'8086, i flag vengono salvati automaticamente dalla CPU, ma esistono significative eccezioni in cui questo non accade. Qualora la CPU non salvasse e ripristinasse automaticamente i flag, sarà il programmatore a doverlo fare, come prima ed ultima istruzione della ISR.

Flag di interrupt

Per ragioni che spiegheremo successivamente, eseguire una ISR a interrupt abilitati può essere "pericoloso".
Dunque l'8086 prima di entrare in una ISR cancella il flag di interrupt, cioè esegue un'istruzione equivalente a **CLI** (**Cl**ear **I**nterrupt flag).

Questa operazione viene eseguita dopo che il registro dei flag è stato salvato nello stack. Per questo quando l'istruzione IRET conclude la ISR essa ripristina automaticamente il flag di interrupt a 1, cioè riabilita le interruzioni.

Le ISR vengono eseguite dall'8086 a interrupt disabilitati, a meno che il programmatore non li abiliti esplicitamente con l'istruzione STI (**Se**t **I**nterrupt) eseguita entro la ISR.

Naturalmente questo implica il fatto che durante l'esecuzione delle interruzioni il sistema è cieco agli interrupt.
Siccome durante la ISR potrebbe arrivare una richiesta d'interruzione importante, da servire subito, le ISR devono essere molto veloci nell'esecuzione, risolvere rapidamente il problema di I/O per cui sono state richieste e ritornare.
In effetti quasi tutte le ISR fanno solo due cose: effettuano l'operazione di I/O richiesta e ne ricopiano il risultato in un buffer. Ad elaborare i dati presenti nel buffer penseranno i programmi "normali", che eseguono ad interrupt abilitati.
Se invece la ISR deve eseguire per molto tempo (indicativamente per più di un ms) allora è necessario abilitare gli interrupt, esponendosi a certi "rischi".
Nella ISR di Figura 1 è mostrato proprio questo caso. Quando il test sull'operazione di I/O dà risultato 0 (punto <5> di Figura 1) è necessario eseguire la procedura "UnPoLunga"[1].
Per non lasciare il sistema d'interruzione disabilitato per troppo tempo, prima della chiamata ad "UnPoLunga" si esegue un'istruzione STI, che abilita le interruzioni. Poi al ritorno si riabilitano con CLI (anche se nel caso specifico non ce ne sarebbe bisogno, dato che si ritorna subito e quindi ci pensa la IRET).

Interrupt e registri
Come già detto nel capitolo precedente, il ritorno da una ISR deve lasciare TUTTI i registri nelle stesse condizioni in cui erano al momento dell'interruzione, altrimenti il programma interrotto non viene danneggiato.
All'inizio delle ISR si devono perciò salvare i valori di tutti i registri che vengono modificati dalla ISR stessa, e ripristinare il loro valore prima di ritornare.
L'8086 non fa alcun salvataggio di registri automatico all'ingresso delle ISR, per cui ci deve pensare il programmatore, come indicato ai punti <6> e <7> della Figura 1.
Alcune CPU, comprese quelle della famiglia X86 dal 286 in poi, hanno particolari istruzioni che salvano tutti i registri nello stack ed altre che li ripristinano tutti. Questa soluzione dà la certezza che non ci saranno problemi nel ritornare al programma interrotto, anche se può rendere il programma meno efficiente, perché i registri che non vengono modificati dalla ISR vengono comunque salvati e ripristinati.

Interrupt vettorizzato 8086
L'8086 dispone di un sofisticato sistema per l'interrupt vettorizzato, ed è in grado di saltare automaticamente ad una qualsiasi di 256 possibili diverse ISR. Ciò significa che, almeno potenzialmente, l'8086 è in grado di riconoscere 256 diversi dispositivi e di iniziare la relativa ISR senza l'esecuzione esplicita di alcuna istruzione di programma.

Identificazione del dispositivo
Il meccanismo con cui la CPU sceglie quale di queste 256 ISR debba essere eseguita è basato sull'identificazione del dispositivo, effettuata dal dispositivo stesso, con la scrittura di un numero sul data bus.
Naturalmente questo numero non può essere scritto dal dispositivo in ogni momento, infatti il data bus viene normalmente usato dalla CPU per ben altri scopi! Ergo per questa operazione deve essere richiesta l'autorizzazione della CPU.
Per dare questa autorizzazione serve un nuovo piedino di control "bus", chiamato, nell'8086, **INTAck** ("**Int**errupt **Ack**nowledge" (conferma di interruzione))[2]. Quando INTAck è alto il dispositivo può scrivere il suo numero di identificazione sul bus.
Dunque in un 8086 al lancio di un interrupt accade questo:
- Il dispositivo alza la linea INTR (l'8086 dispone solo di questo piedino per tutte le richiesta d'interruzione "normali")
- La CPU alza il segnale INTAck se IF = 1, se invece IF = 0 essa non dà nessun acknowledge e l'interrupt rimane "pendente" ("pending interrupt"), poi prosegue nel fetch dell'istruzione successiva, ignorando la richiesta d'interruzione
- Il dispositivo scrive il suo numero di identificazione, che d'ora in poi chiameremo **INT#** [3], nei primi otto bit del data bus (il numero è minore di 256)
- La CPU legge l'INT# dal data bus e lo usa per l'accesso alla ISR del dispositivo.

Per semplificare l'elettronica dei dispositivi di I/O di solito viene interposto fra il dispositivo di I/O ed il sistema d'interruzione della CPU un particolare circuito integrato che fa da interfaccia fra i due, realizzando il .
Questo dispositivo viene detto "**interrupt controller**", quello più comune con l'8086 è l'8259, usato nel primo PC IBM; esso verrà trattato in dettaglio nel seguito.
Generalmente un interrupt controller è in grado di raggruppare diverse linee d'interruzione e di passare le loro richieste ad un unico piedino d'interruzione della CPU, come indicato in Figura 2 da IRQ 1, IRQ 2, .. . La sigla IRQ significa **I**nterrupt **Req**uest.

1 Come vedremo anche la chiamata a procedure deve essere effettuata con cautela nelle ISR, la procedura di Figura 1 è quindi un esempio di ISR piuttosto "difficile", anche visto il suo nome "UnPoLunga"!
2 Questo segnale viene alzato per ogni richiesta d'interruzione, proveniente da qualsiasi dispositivo, per cui non può giungere ad un solo dispositivo specifico e non è utile per segnalare ad un dispositivo che la sua richiesta è stata servita. Ciò significa che non può essere usato per risolvere il problema del riconoscimento dell'interruzione, che abbiamo trattato nel capitolo precedente.
3 Il simbolo di diesis (#), chiamato "pound" o "sharp" dagli Americani, viene da loro spesso usato al posto della parola "number" od al posto di un numero. Quindi leggeremo INT# come "INT number".

Un altro compito del controllore d'interruzione è regolare la priorità degli interrupt che gestisce, facendo in modo che, in caso di IRQ concomitanti, venga lanciato per primo l'interrupt più importante.
L'ultima importante funzione dell'interrupt controller è quella di "mascherare" alcune sorgenti d'interruzione. Manipolando opportunamente i bit di un registro del controllore si è in grado di abilitare o disabilitare alcuni specifici interrupt, per i quali il controllore non passerà alla CPU le eventuali richieste di interruzione.

!!!!

Figura 2: richiesta e conferma di interrupt

Tabella dei vettori d'interruzione
La fase successiva è il salto alla ISR del dispositivo che ha richiesto l'interruzione.
Il salto è eseguito in modo indiretto, leggendo in memoria l'indirizzo della ISR.
Dato che le ISR possono essere 256 devono esistere in memoria 256 locazioni "fisse" nelle quali si può scrivere l'indirizzo di ciascuna delle ISR.
Gli indirizzi delle ISR sono contenuti nella "**tabella dei vettori d'interruzione**". Il primo chilobyte della memoria di un 8086 è destinato a questa tabella, che contiene 256 indirizzi segmentati. (32 bit per ogni indirizzo, 256 * 32 bit = 1 Kbyte).
Ciascuno di questi indirizzi punta alla prima istruzione di una ISR e viene detto **vettore d'interruzione**.
Il vettore d'interruzione dell'INT 1 sta all'indirizzo zero e prende 4 byte, il vettore dell'INT 2 sta all'indirizzo 4, in generale il vettore dell'INT n si trova all'indirizzo (n* 4).
La sequenza d'interruzione si può ora completare con il salto alla ISR. La CPU ha già letto l'INT#, o lo usa per calcolare l'indirizzo nella tabella dei vettori al quale si trova l'indirizzo della ISR.
Il calcolo è semplice: all'indirizzo INT# * 4 si trova l'offset dell'indirizzo, da copiare in IP, a [INT# * 4 + 2] il segmento, da copiare in CS.

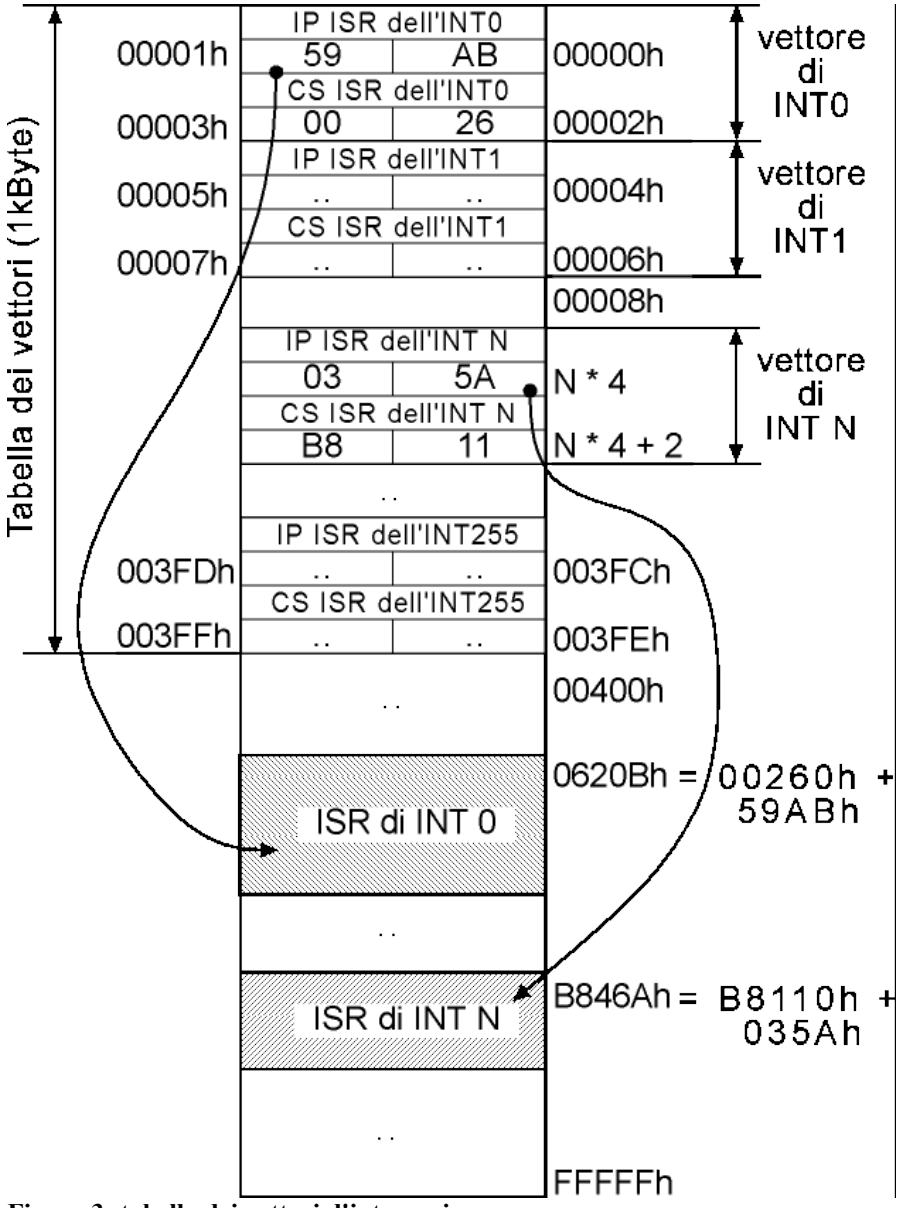

Figura 3: tabella dei vettori d'interruzione

Sequenza di interrupt dell'8086

Per completare la sequenza di interrupt manca il salvataggio del "Program Counter", che fino a qui abbiamo trascurato. ,
In un 8086 esso consiste nella scrittura nello stack di CS e di IP, nell'ordine, in modo che il valore che "emerge" dallo stack sia la "parte bassa" dell'indirizzo (CPU little endian).

A questo punto abbiamo tutti gli elementi per elencare nel dettaglio tutti i passi di una sequenza d'interruzione dell'8086.

<FILE>
SequINT.FH5
</FILE>

Figura 4: sequenza di interrupt di un 8086

(1) **Dispositivo**. Per richiedere servizio da parte della CPU alza il suo segnale di richiesta d'interruzione, che è collegato all'interrupt controller
(2) **Interrupt controller**. Alza il segnale di richiesta d'interruzione della CPU (INTR)
(3) **CPU**. Completa la fase di execute dell'istruzione corrente
(4) **CPU**. Verifica se c'è una richiesta di interruzione su INTR. Se IF = 0 oppure INTR = 0 passa al fetch dell'istruzione successiva del programma correntemente in esecuzione, non eseguendo alcuna ISR
(5) **CPU**. Se IF = 1 e (and) INTR = 1 salva nello stack i flag (equivalente a "PUSHF")
(6) **CPU**. Mette a zero l'interrupt flag (equivalente a "CLI")
(7) **CPU**. Salva nello stack CS (equivalente a "PUSH CS")
(8) **CPU**. Salva nello stack IP (equivalente a "PUSH IP")
(9) **CPU**. Dà la conferma d'interruzione all'interrupt controller, alzando la linea INTack

(10) **Interrupt controller**. Mette sul data bus il numero di vettore del dispositivo che sta interrompendo (INT#)
(11) **CPU**. Legge l'INT# dalle linee 0-7 del data bus
(12) **CPU**. Scrive in IP il valore (a word) letto in memoria all'indirizzo INT# * 4 (come "MOV CS , 0: [INT# * 4]")
(13) **CPU**. Scrive in CS il valore letto in memoria all'indirizzo INT# * 4 + 2 (come "MOV CS , 0: [INT# * 4 + 2]")

questo conclude la sequenza di interrupt. La prossima fase di fetch viene eseguita all'indirizzo CS:IP, per cui si salta alla prima istruzione della giusta ISR.

Figura 5: istruzioni equivalenti di una sequenza di interrupt e di IRET

1.1.1 Uso "software" delle sequenze di interrupt

I circuiti della CPU 8086 che servono per realizzare la sequenza di interrupt sono usati anche per gli interrupt software e per le "eccezioni".

Interrupt software

L'interrupt software è un modo per realizzare chiamate indirette a procedure, in modo più flessibile rispetto all'istruzione CALL.
Per chiamare un interrupt software si usa l'istruzione INT, che ha la seguente sintassi:

```
INT <Numero di vettore>
```

Esempio:

```
INT 21h ; esegue la routine di risposta all'interruzione dell'INT 21h
```

Il funzionamento di INT è molto semplice: viene eseguita una sequenza di interruzione, che comprende molti dei passi di quella dell'interrupt hardware, descritta nel paragrafo precedente.
Due sole, ma fondamentali, le differenze rispetto all'interrupt hardware:
1. il numero di vettore non viene prelevato dal data bus, ma dal codice stesso del programma (è <Numero di vettore>). Questo implica che non c'è lo scambio INTR – INTAck ed il controllore d'interruzione non è interessato dalla sequenza
2. Il flag di interrupt viene ignorato e la sequenza eseguita comunque (sarebbe assurdo pensare che la CPU non esegua l'istruzione INT se IF = 0; se così fosse cosa dovrebbe fare? Si dovrebbe bloccare per sempre, ignorare l'istruzione o cos'altro?).

Dal punto di vista funzionale un interrupt software non è un interrupt, perché non "interrompe" nulla, ma è piuttosto una procedura con chiamata indiretta.
Il fatto però che esegua una sequenza quasi identica ad un interrupt significa che vengono salvati nello stack CS, IP ed il registro del flag, per cui per tornare al programma principale da un interrupt software bisogna usare IRET.
Il meccanismo dell'interrupt software è flessibile ed efficiente; per questo motivo viene usato come metodo per effettuare le chiamate al Sistema Operativo, non solo in MS-DOS (INT 21h), ma anche in Linux.
Naturalmente un vettore d'interruzione usato per un interrupt software non potrà essere usato anche per un interrupt hardware. Dunque dei 256 possibili vettori d'interruzione solo alcuni sono destinati a interrupt hardware (nel PC sono solo 16). Gli altri vettori possono essere usati dal software di sistema (Sistema Operativo), dalle eccezioni della CPU ed eventualmente, i rimanenti, anche dai programmi d'utente.

Eccezioni della CPU

In occasione di particolari eventi, che richiedono un trattamento immediato da parte del Sistema Operativo e/o dai programmi applicativi, la CPU inizia autonomamente un processo di interruzione.
Alcuni vettori del sistema di interruzione sono perciò riservati alla CPU, che li usa per lanciare le ISR relative a questi eventi. Nell'architettura degli X86 i primi 32 vettori d'interruzione sono riservati alle eccezioni della CPU.
Questi "interrupt", che non vengono lanciati né dall'hardware, né dal software, ma autonomamente dalla CPU, vengono detti **eccezioni** (exceptions) e scattano quando la CPU si trova in alcune condizioni anomale, che il progettista della CPU ha stabilito di segnalare in questo modo.
Le due differenze delle eccezioni rispetto all'interrupt hardware sono:
1. Il numero di vettore non si può cambiare ed è stabilito dal progettista della CPU. Ogni condizione che genera un'eccezione ha il suo vettore d'interruzione, che viene pubblicato nel manuale della CPU.
2. L'interrupt flag viene ignorato

Segue una tabella con le eccezioni dell'8086:

INT #	Commento
INT 0	Errori nella divisione intera (overflow, divisione per zero)
INT 1	Single step: interrompe l'esecuzione temporaneamente, quando il flag T (trap) è a 1 (usato dai debugger)
INT 2	NMI (not maskable interrupt): viene generata da una transizione della tensione sul piedino NMI della CPU
INT 3	Lanciata dall'istruzione INTO, che ha codice operativo di 1 Byte (usata per i breakpoint dal debugger software)
INT 4	Interrupt on overflow. Viene lanciato dall'istruzione INTO, però solo nel caso in cui il flag di overflow sia ON. Usato per avere un'eccezione anche nel caso di errori aritmetici in somma o sottrazione.

Figura 6: i principali vettori d'interruzione di un PC MSDOS.

Esempio:
Il vettore d'interruzione zero (INT 0) è riservato, per decisione del costruttore, all'errore di divisione per zero.
Quando durante un programma si esegue un'istruzione DIV con divisore nullo parte automaticamente l'ISR dell'INT 0 (quella il cui vettore si trova all'indirizzo 0).

Nelle CPU successive della serie 80X86 le eccezioni sono molto più numerose, perché devono comunicare i molti tipi di errore possibili in quelle CPU sofisticate, che hanno funzionalità per: la "protezione della memoria", la memoria virtuale, coprocessori aritmetici interni, cache interne.
Nell'Appendice A1 sono indicati i principali vettori d'interruzione, sia hardware che software, per un PC su cui sia installato MS-DOS.

Il sistema d'interruzione nella famiglia X86

Nell'80286 (286, CPU da 16 bit) il sistema d'interruzione è analogo a quello dell'8086, vengono aggiunte nuove eccezioni, relative alle condizioni anomale riscontrate quando la CPU lavora in modo protetto.
Dall'80386 (386, CPU da 32 bit) in poi il sistema d'interruzione è cambiato in modo significativo e rimane identico anche nei Pentium. La tabella dei vettori d'interruzione è stata "virtualizzata" e può ora trovarsi fisicamente in qualsiasi locazione di memoria, anche se per i vecchi programmi, scritti per le CPU precedenti al 386, tutto può funzionare come se fosse ancora nel primo kByte della memoria.
Dato che per comprendere il funzionamento degli interrupt in modo protetto è necessario introdurre prima le tecniche usate dalla CPU per supportare la memoria virtuale, si posticipa la sua descrizione più dettagliata al capitolo ???? "Gestione della memoria", paragrafo "Interrupt ed eccezioni in modo protetto".

Eccezioni X86
Le eccezioni, dal 286 in poi, possono mettere nello stack un codice che identifica la causa dell'errore e può aiutare il software a decidere come trattarlo. In modo reale non viene mai passato alcun codice nello stack.
Intel distingue fra tre tipi di interruzioni non hardware: trap, fault e abort.
Le **trap** ("trappole") sono simili a interrupt software (anzi, si può dire che gli interrupt software sono delle trap).
Le trap partono dietro richiesta del software oppure in risposta ad un evento interno che richiede l'esecuzione di una "normale" procedura, in modo analogo ad un interrupt hardware.
Per questo al ritorno da una trap (alla IRET) l'istruzione che viene eseguita è sempre quella successiva a quella in cui è avvenuta la chiamata.
I **fault** (errori) sono generati dalla CPU in conseguenza di un evento assimilabile ad un errore. La routine di risposta all'eccezione di fault fa concludere il programma oppure rimedia all'errore.
Dato che l'errore è rimediato se si fa la IRET essa riporta alla stessa istruzione che aveva generato l'errore. Il programma, "aggiustato", ora potrà funzionare regolarmente.
Gli **abort** sono invece errori gravi, che devono far interrompere la elaborazione in ogni caso. Nel caso di un abort il programmatore non deve far ripartire l'elaborazione del programma che l'ha generata (cioè non deve fare la IRET).
Le trap non mettono mai nessun codice d'errore nello stack, i fault lo fanno solo qualche volta, gli abort sempre.
Nell'Appendice A1 si può trovare una tabella che elenca tutte le eccezioni delle CPU 80X86.
I dettagli sulle eccezioni e sul sistema d'interruzione delle CPU più moderne della famiglia X86 saranno dati in seguito

1.1.2 Interrupt non mascherabile

Le CPU della famiglia X86 hanno un piedino che serve per generare un tipo di interrupt la cui ISR viene eseguita comunque, anche se il sistema delle interruzioni è disabilitato. Questo piedino è chiamato NMI (**N**ot **M**askable **I**nterrupt).
La ISR dell'interrupt non mascherabile risiede al vettore INT 2. L'esecuzione di una INT 2 in risposta ad un segnale NMI non può essere interrotta in alcun modo.

1.1.3 8259

L'8259 è un circuito di I/O di una certa complessità, sviluppato specificamente per microprocessori della famiglia X86, la cui funzione è semplificare l'accesso al sistema d'interruzione.
Esso accetta in ingresso 8 linee di interrupt, da 8 diversi dispositivi, ed è in grado di interrompere una CPU 8086, rispettando la sua sequenza di interrupt e producendo sul data bus un numero diverso per ciascuno degli interrupt che accetta in ingresso.
L'8259 è perciò un "interrupt controller", le cui caratteristiche funzionali possono essere cambiate scrivendo in alcuni suoi registri. Per questo è stato denominato "**P**rogrammable **I**nterrupt **C**ontroller" (**PIC**)[4].
Si tenga dunque presente che quando si legge PIC bisogna far distinzione fra l'interrupt controller ed il microcontrollore.

Le funzioni del PIC sono:
- gestire 8 interrupt separati con un'unica linea INTR al microprocessore
- realizzare una priorità fra le 8 sorgenti d'interruzione
- gestire la sequenza INTR – INTACK per conto dei dispositivi
- identificare la sorgente d'interruzione scrivendo l'INT # sul data bus
- predisporre il funzionamento "master– slave" di altri PIC (vedi in seguito)

Per realizzare le sue funzioni un 8259 dispone di 8 linee di ingresso collegate direttamente ai dispositivi che devono lanciare un'interruzione.

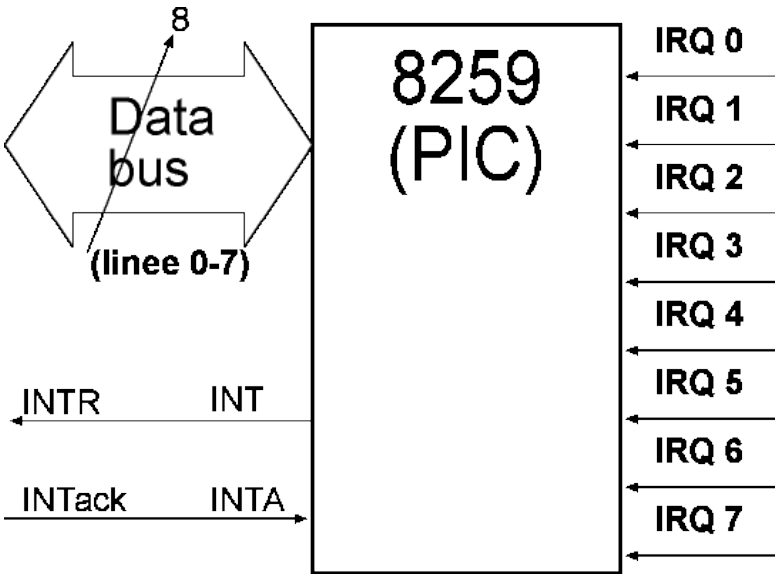

Figura 7: Gli ingressi e le uscite più significative di un 8259

Chiameremo queste linee "**IRQ**", cioè "**I**nterrupt **Req**uest" (richiesta d'interruzione), e le identificheremo con un numero da zero a sette (IRQ0, IRQ1, .. , IRQ7).
In Figura 7 sono indicate le linee di IRQ con accanto un simbolo che corrisponde al dispositivo collegato a quell'IRQ nell'architettura dei PC IBM compatibili.
Dal lato CPU un 8259 ha le linee, già descritte, di INTR e INTACK, per poter sapere quando ha l'autorizzazione per identificare il dispositivo, scrivendo il suo INT # sul data bus.

Architettura interna di un 8259

Al suo interno un 8259 è costituito di poche unità funzionali, illustrate in Figura 8.

4 Il "nome" PIC è anche usato dall'azienda "Microchip" per denominare una famiglia di microcontrollori di bassissimo costo, impiegati in molte applicazioni "embedded". Nel caso di questo microcontrollore la sigla PIC significa "Peripheral Interface Controller". Questo PIC non ha dunque nulla a che fare con il controllore d'interruzione del quale si tratta qui.

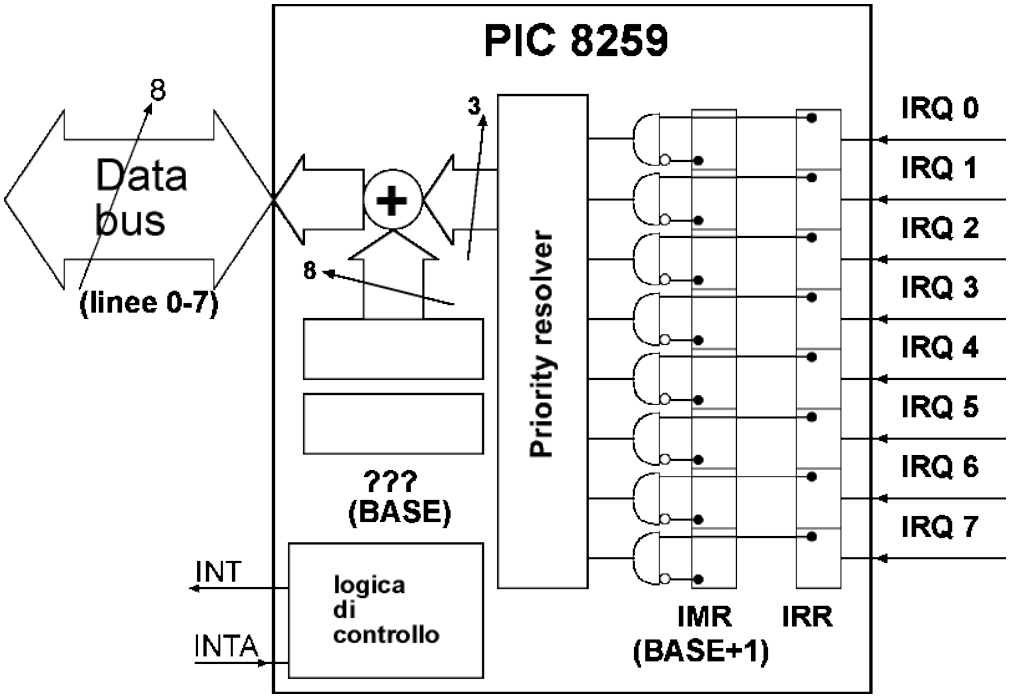

Figura 8: schema di principio di un 8259

Le richieste di interruzione vengono "raccolte" da un buffer collegato con l'esterno, detto "IRR", che "copia" al suo interno i segnali provenienti dai dispositivi.

Queste richieste possono passare allo stadio successivo, indicato come "priority resolver" (discriminatore di priorità) in Figura 8, solo se sono abilitate dal registro IMR (**I**nterrupt **M**ask **R**egister).

Come indicato dalle porte AND della figura, una richiesta d'interruzione può passare al priority resolver solo se il corrispondente bit del registro IMR ha valore 0 (notare che un ogni AND c'è un pallino di negazione per le linee provenienti dall'IMR).

Dunque la funzione dell'IMR è quella di lasciar passare solo gli interrupt il cui bit è a zero.

Per "mascherare" una sorgente d'interruzione basta scrivere un uno nel bit corrispondente. In questo caso l'8259 non alza INTR, anche se il dispositivo alza la sua IRQ.

Nel modificare IMR bisogna fare molta attenzione a cambiare solo il bit che interessa, senza influenzare gli altri bit, perché altrimenti tutto il sistema rischia di bloccarsi.

Esempio:

```
; istruzioni che mascherano IRQ3:
CLI
IN AL, IndirizzoIMR  ; funziona se l'indirizzo < 256
OR AL, 1000b     ; maschera per mettere 1 nel bit di peso 3
OUT IndirizzoIMR, AL
STI
```

Il priority resolver è l'unità che ha il compito di stabilire quale fra le richieste di interrupt attive in un certo istante è la più importante.

Il criterio per stabilire l'interrupt più importante può essere diverso in base a come il dispositivo viene programmato.

In un PC e con i Sistemi Operativi più usati il circuito è programmato per fare in modo che l'interrupt più importante sia quello che ha il numero più basso. Perciò se i dispositivi lanciano contemporaneamente IRQ3, IRQ1 e IRQ7 la prima ISR ad essere eseguita sarà quella che corrisponde a IRQ1.

Il priority resolver realizza elettricamente la semplice regola appena enunciata fa uscire il numero di IRQ che deve essere richiesto.

La fase successiva è la generazione del vettore d'interruzione che corrisponde all'IRQ prescelto.

Il numero di INT si ottiene sommando il numero contenuto nel registro !!!!NOME al numero di IRQ che è passato dal priority resolver. Il risultato viene messo sul data bus nel momento in cui la CPU alza INTACK.

Comando EOI (End Of Interrupt)

Per poter funzionare il priority resolver deve disabilitare nel PIC tutti gli interrupt di priorità pari od inferiore a quello richiesto alla CPU.

Perché gli interrupt riprendano a funzionare deve essere emesso un comando "**E**nd **o**f **I**nterrupt" (**EOI**).

L'8259 può essere programmato per funzionare con un EOI automatico. Peraltro nel S.O. MS-DOS ciò non avviene, per cui le ISR devono farlo esplicitamente durante la ISR.

E' perciò necessario che la CPU (il programmatore) comunichi al PIC che l'interrupt che esso aveva lanciato è stato servito, in modo che esso possa riabilitare gli interrupt meno prioritari.
Ciò si ottiene emettendo il comando "end of Interrupt", cioè scrivendo il numero 20h all'indirizzo base del PIC.

Gli 8259 nei PC IBM e compatibili

Il primo personal computer IBM, il PC XT, fu il capostipite dei PC odierni e conteneva un solo PIC 8259, per cui poteva gestire un massimo di 8 sorgenti d'interruzione hardware.
L'unico PIC di un PC XT è mappato in I/O all'indirizzo base 20h (32d). L'indirizzo di I/O dell'IMR è perciò BASE + 1 = 21h.
I dispositivi cablati ai piedini dell'8259 sono, in un PC XT, quelli indicati nella Tabella 1:

IRQ #	INT #	Funzione
IRQ0	08h	System timer
IRQ1	09h	Tastiera
IRQ2	0Ah	Riservato
IRQ3	0Bh	Porta seriale COM2
IRQ4	0Ch	Porta seriale COM1
IRQ5	0Dh	Disco rigido
IRQ6	0Eh	Controller floppy disk
IRQ7	0Fh	Stampante LPT1

Tabella 1: IRQ in un PC XT

Si può arguire dalla tabella che il numero di vettore (INT#) del primo IRQ dell'8256 è 8.
SI nota anche che non rimane nessun interrupt libero, dato che IRQ2 è riservato. Per eventuali schede aggiuntive che utilizzavano l'interrupt bisognava usare quello di uno dei dispositivi di tabella[5], oppure condividere una linea di interrupt fra più dispositivi, complicando il software di ISR.
E' chiaro quindi che c'era la necessità di espandere il numero di interrupt gestiti da un PC e ciò è quanto venne fatto al momento della progettazione del successore del PC XT, che prese il nome di PC AT (Advanced Technology).
Nel PC AT si sfruttava quella caratteristica dell'8259 che non abbiamo ancora illustrato, cioè la possibilità di collegare i PIC in modo "master– slave".
In un PC AT sono presenti due 8259, collegati fra loro e con la CPU in modo da mantenere la completa compatibilità con il PC XT.
Uno dei due PIC, detto "master PIC" (padrone), è collegato esattamente come l'unico PIC del PC XT.
Il secondo PIC, detto "slave PIC" (schiavo), è collegato in modo da lanciare in suo interrupt non sulla linea INTR della CPU, ma come IRQ numero 2 del PIC master (vedi Figura 9)

5 storicamente fu usato quello della porta parallela, dato che non funzionava correttamente nella scheda originale IBM, oppure la seriale COM2, perché erano pochi in computer su cui erano presenti più di una porta seriale.

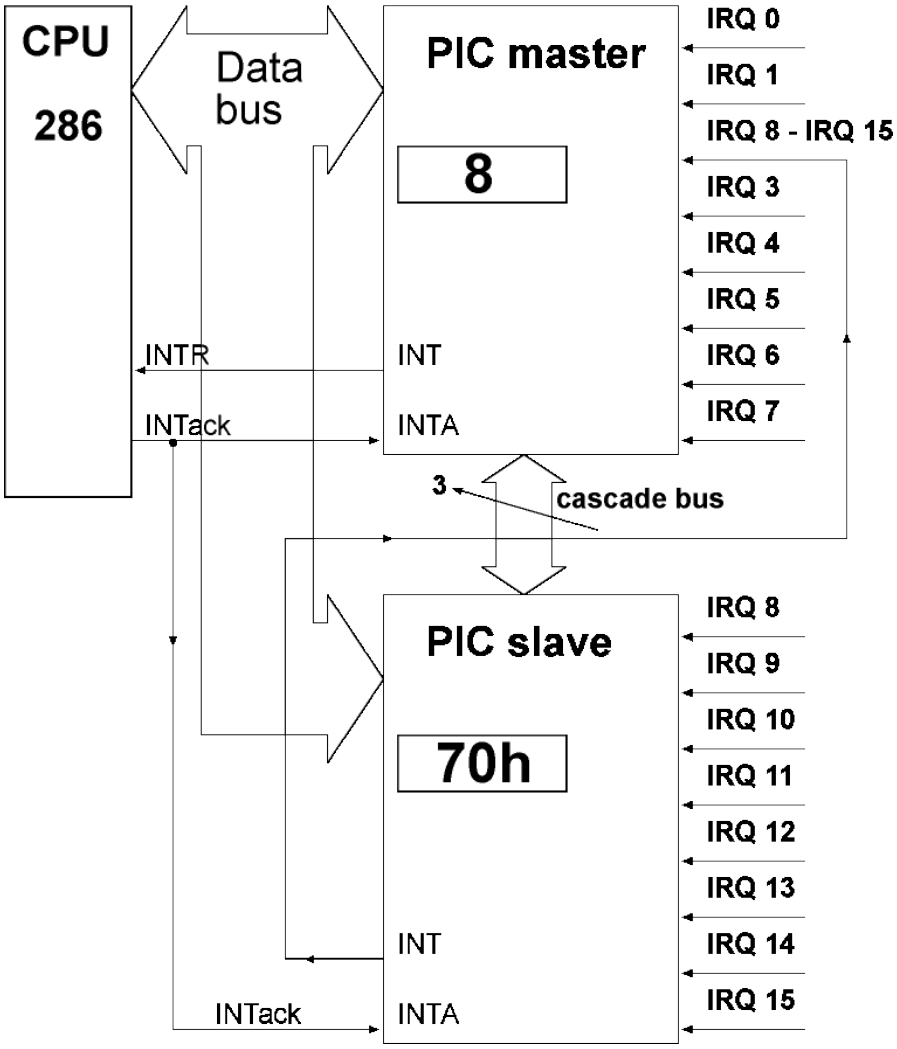

Figura 9: PIC master e slave

Questo significa che l'IRQ2, prima tenuto riservato, nel PC AT è lanciato ogni volta che c'è una richiesta d'interruzione da uno qualsiasi dei device collegati al PIC slave.
Come indicato in Figura 9 nel PC AT si chiamano IRQ8, 9, .. 15 le linee di richiesta d'interruzione collegate al PCI slave.
Data la modalità di funzionamento degli 8259 ed il collegamento master–slave, la priorità delle IRQ di un PC AT non può che essere la seguente: IRQ0, IRQ1, IRQ8, IRQ9, .. , IRQ3, IRQ4, .. IRQ7.
Il PIC slave di un PC AT è mappato in I/O all'indirizzo BASE = A0h (160d), per cui l'IMR del PIC slave è all'indirizzo A1h.

	PIC Master		PIC slave	
Ind. base (per EOI)	20h	32d	A0h	160d
IMR	21h	33d	A1h	161d
INT # del primo IRQ del chip	08h	08d	70h	112d

Tabella 2: indirizzi dei registri dei PIC master e slave in un PC AT e numero di vettore della prima IRQ

Con la presenza di due interrupt controller le sorgenti di interruzione a disposizione divengono 15, dato che una va "sprecata" per la ridirezione dell'IRQ2.
Molti interrupt rimasero liberi per l'assegnazione a schede di espansione. Nel corso del tempo questi interrupt sono stati occupati da periferiche che inizialmente non erano presenti e che sono diventate di uso comune successivamente (es. schede di rete).

IRQ #	PIC	INT #	Funzione
IRQ 0	master	08h	System timer
IRQ 1	master	09h	Tastiera
IRQ 2	master	0Ah	Riservato nel PC XT, IRQ ridiretto dal PIC slave nel PC AT
IRQ 8	slave	70h	CMOS real time clock
IRQ 9	slave	71h	???? !!!!
IRQ 10	slave	72h	non assegnato dall'IBM nel PC AT

IRQ 11	slave	73h	non assegnato dall'IBM nel PC AT
IRQ 12	slave	74h	non assegnato dall'IBM nel PC AT
IRQ 13	slave	75h	Coprocessore aritmetico (nel PC AT un 287, da 486 in poi è interno alla CPU)
IRQ 14	slave	76h	controller del disco rigido
IRQ 15	slave	77h	non assegnato dall'IBM nel PC AT
IRQ 3	master	0Bh	Interfaccia seriale COM 1
IRQ 4	master	0Ch	Interfaccia seriale COM 2
IRQ 5	master	0Dh	Interfaccia parallela LPT2
IRQ 6	master	0Eh	controller floppy disk
IRQ 7	master	0Fh	Interfaccia parallela LPT1

Tabella 3: Tutti i vettori d'interruzione hardware del PC, in ordine di priorità

Il numero di vettore d'interruzione (INT#) sul data bus viene scritto dal PIC interessato, non dal master. Per poter sapere se deve scrivere sul data bus un PIC slave comunica con il master attraverso un bus di tre linee, detto "cascade bus". Per quanto nel PC sia utilizzato un solo slave, un 8259 master ne può controllare fino ad 8.
L'8259 può essere programmato per funzionare in 4 modi diversi. Il "modo 0" è quello di default e viene usato nei PC; gli altri modi hanno i seguenti nomi:
- rotating priority
- special mask mode
- rolled mode

Il comando EOI nei PC AT

Se il numero di IRQ è compreso fra 0 e 7, IRQ2 esclusa, l'unico PIC interessato all'interruzione è il master. In questo caso è necessario spedire il comando EOI soltanto al PIC master. Nel caso invece di interrupt fra 8 e 15 sono coinvolti entrambi i PIC, per cui ad entrambi bisogna mandare il comando EOI.
Perciò, se l'IRQ cui si sta rispondendo è fra 0 e 7, bisognerà fare una OUT di 20h solo all'indirizzo 20h, mentre se è fra 8 e 15 bisognerà farlo anche in A0h.
Esempio:

```
IRQ12:
    ..
    MOV AL, 20h
    ; EOI per il PIC master:
    OUT 20h, AL
    ; EOI per il PIC slave:
    OUT A0h, AL
    ..
```

L'inizializzazione dei due PIC viene fatta da parte del Sistema Operativo nel momento del bootstrap, è chiaro che il software d'utente non dovrà cambiare né la modalità di funzionamento dei PIC, né il numero di vettore che essi mettono sul data bus (INT#), pena l'irrimediabile blocco del sistema.

~~Processor Family Developer's Manual, Volume 3: Architecture and Programming Manual"~~
~~Advanced Micro Devices (AMD) "AMD-K5~~