

LEZIONE 21/10/08

Debugging

L'operazione di debug viene ottenuta attraverso l'utilizzo di interrupt sia software che hardware. Nei microprocessori recenti (quelli in cui viene gestita la memoria virtuale) la gestione del debug risulta essere difficoltosa e richiede la presenza di hardware apposito, di seguito viene analizzata solo nel caso di un sistema in real-mode.

Le funzioni principali di debug risultano essere:

- inserimento di breakpoint: durante l'esecuzione del programma da debuggare, nel caso venga raggiunta una ben definita istruzione, il controllo passa al debugger;
- esecuzione step by step: il controllo del programma passa istruzione per istruzione al debugger.

La funzione di step by step è attivata tramite il flag TF: l'unità di controllo del processore, al termine di ogni istruzione ne controlla lo stato, nel caso in cui sia stato settato viene generata una trap (corrispondente ad un INT 4); il debugger deve aver precedentemente posizionato nella IVT, alla posizione 4, una routine del debugger stesso per gestire tale modalità, inoltre lo stesso deve anche farsi carico di salvare l'indirizzo dell'istruzione sospesa, così che sia possibile riprendere l'esecuzione da tale riga di codice.

Per quanto riguarda la realizzazione della funzione di breakpoint, si potrebbe inserire all'interno del debugger una tabella contenente tutti gli indirizzi di breakpoint. Mentre il programma viene eseguito, il debugger dovrebbe confrontare il valore del program counter con tutti gli indirizzi memorizzati in tabella ed eventualmente chiamare la funzione di breakpoint.

Una soluzione più efficiente consiste nel sostituire ad ogni istruzione su cui è stato impostato un breakpoint con una istruzione di salto incondizionato (JMP) ad una apposita routine del debugger: tale routine a sua volta andrà a sostituire al posto del JMP l'istruzione originale, in modo tale che possa venire successivamente eseguita (nel caso in cui sia un breakpoint "persistente", viene anche attivata la modalità step-by-step, in modo del tutto trasparente all'utente, così quando viene eseguita l'istruzione successiva al breakpoint viene richiamato il debugger il quale ha la possibilità di andare a risostituire l'istruzione del breakpoint con un jump).

Vi sono alcune considerazioni da fare riguardo i breakpoint:

- Il programma deve essere caricato in RAM, se fosse in ROM non potrei andare a sostituire le istruzioni (ad esempio il bios, per debuggarlo bisogna prima caricare in RAM la parte da modificare),
- L'istruzione originaria è stata sostituita da un salto ma deve essere stata precedentemente salvata dal debugger da qualche parte,
- Le istruzioni hanno lunghezza variabile, saranno quindi necessari dei JMP "particolari" in modo tale che non vadano a sovrascrivere le istruzioni successive a quella su cui si vuole applicare il breakpoint.

Per quanto riguarda l'ultimo punto, si veda l'esempio seguente:

- 1) MOV AX,[BX] (3 byte)
- 2) ADD AX, CX (1 byte)
- 3) NOP (1 byte)
- 4) MOV AX,[--] (3 byte)

In questo caso sostituendo la Nop con una JMP (la quale sarà sicuramente di tipo FAR e quindi occuperà 5 byte), l'istruzione 4 verrebbe sovrascritta determinando il malfunzionamento del programma. Vi è quindi la necessità di poter effettuare dei salti incondizionati di tipo far utilizzando istruzioni che siano lunghe 1 byte (in tal modo vi è la certezza di non andare a sovrascrivere altre righe di codice): si potrebbe utilizzare una INT n (con n che varia da zero a 255 e quindi occupa 1

byte) ma in generale sarebbe sempre su 2 byte. La soluzione si è trovata nell'utilizzo dell'istruzione INT 3 la quale è codificata su un solo byte: essa è una INT particolare ed è utilizzata per fare un salto incondizionato ad una routine del debugger, che compierà tutte le operazioni del caso; naturalmente il debugger avrà provveduto a scrivere nella posizione 3 del vettore delle istruzioni l'indirizzo a cui si trova la routine che deve essere eseguita.

SLIDE 23: contesto di interrupt vettorializzato e centralizzato, tutte le richieste dei periferici convergono al PIC (Programmable Interrupt Controller, 8259 nell'8086) o all'APIC (Advanced PIC, nel caso di sistemi multi-core, dove il problema è più complesso). Il segnale INTA corrisponde all'acknowledgment, l'A0 serve per programmare il PIC.

Il processore termina l'esecuzione dell'operazione e solo dopo riconoscerà il segnale di interrupt attivo.

Vi sono molte problematiche legate agli interrupt, in particolar modo nelle architetture di ultima generazione, dove vi sono pipeline e multi-core: cosa accade con una pipeline? Alcune istruzioni saranno quasi finite, altre appena iniziate, caricherà l'interrupt nella pipeline, con quale istruzione mi fermo? Svuoto la pipeline? La accodo alla fine della pipeline? E se nella pipeline avevo una "clear interrupt"?

SLIDE 27: è rappresentata una tempistica migliore di quello che accade.

SLIDE 29: vi è un Interrupt Request 3 che ha una priorità maggiore rispetto ad IR 5 e quindi viene servito prima anche se in realtà è stato attivato solo successivamente; di norma in seguito viene servito l'IR 5 ma non è detto.

SLIDE 31: tempo di latenza calcolato come somma di tre fattori:

- t_i , l'istruzione più lenta di cui bisogna aspettare la terminazione (sottovalutiamo il caso comportato dalla presenza di una pipeline), dobbiamo conoscere le performance del sistema e quindi l'architettura, non ha importanza la frequenza di clock della CPU, noi stiamo facendo una analisi del caso peggiore e questo sarà causato da istruzioni dipendenti dalla periferia;
- t_{hw} , accesso alla memoria per leggere l'IVT e salvare i valori nella cache, il tempo dipenderà dal parallelismo del bus, se i valori sono in cache oppure in ram (questo ultimo è il caso peggiore, consideriamo questo);
- t_r , la routine di gestione dell'interrupt non va subito a leggere la periferica, prima deve compiere delle istruzioni che ci impiegano del tempo (appunto questo ultimo) per poter essere effettuate.

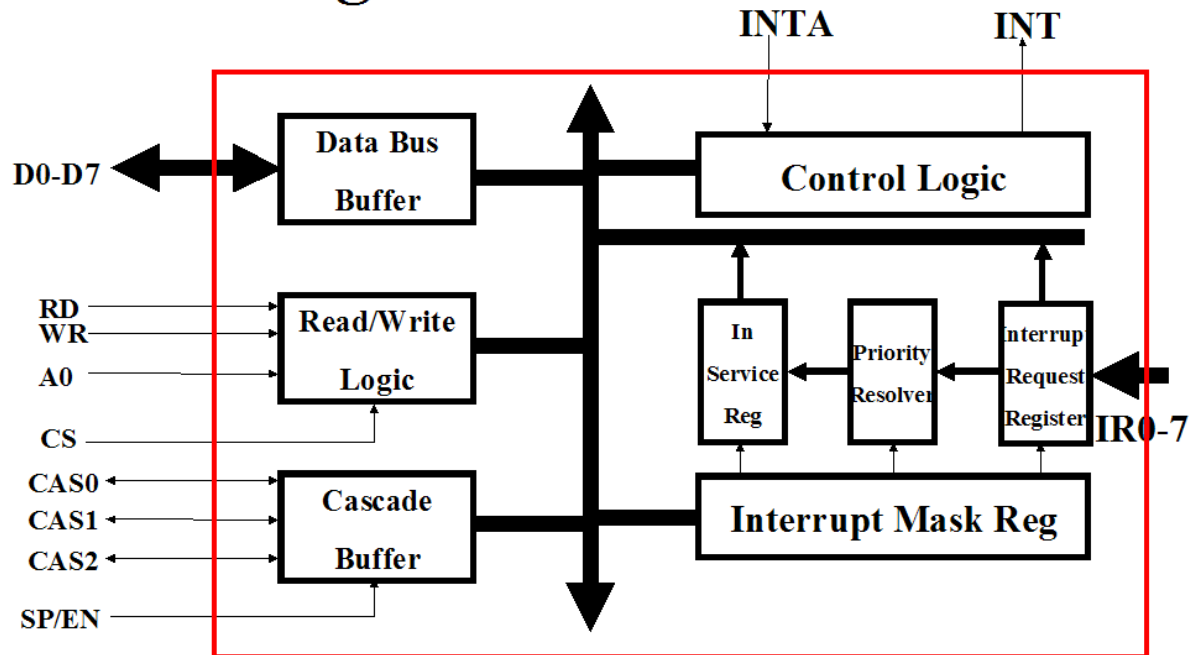
Il tempo di latenza deve essere minore del transfer rate, altrimenti il processore non riesce a servire i vari interrupt, inoltre anche il tempo di gestione deve essere minore del transfer rate, così da poter permettere alla CPU di svolgere le altre operazioni del sistema.

Scrivere ISR richiede una particolare attenzione, le prestazioni generali del sistema non sono affatto trascurabili, influiscono molto sul tempo di latenza.

INTERRUPT: 8259/APIC PC interrupts

L'8259 è un chip (attualmente lo si trova incorporato nel South bridge mentre in passato si trattava di un chip indipendente) costituito da 8 pins di dato (D0-7), da un pin di read (RD), un pin di write (WR), un pin di chip select (CS) e 3 pin per la gestione dei segnali di CAS (CAS0-2) tramite i quali è possibile gestire l'interconnessione di più controller 8259 posti in cascata, fino ad un massimo di 64 livelli di interrupt..

Diagramma a Blocchi



Prof. Marco Mezzalama 08/09

L'Interrupt Request Register (IRR) riceve tutti i segnali di richiesta di interrupt (IR0-7). In caso di richiesta di interrupt, viene settato il corrispondente bit dell'IRR. A questo punto l'8259 invia un segnale di INT alla CPU e attende la risposta (INTA). Nel caso ci fossero più richieste contemporaneamente verrebbero attivati più bit tra i quali viene selezionato quello con priorità più alta. Risulta, quindi, di fondamentale importanza l'uso del priority resolver che seleziona il bit associato all'IR_n con priorità maggiore. A questo punto l'interrupt con priorità più alta viene memorizzato nell' in service register (ISR) abilitando il bit corrispondente e resettando il corrispondente bit del registro IRR.

L'Interrupt Mask Register (IMR) contiene una maschera da applicare all'IRR in modo tale da abilitare o mascherare eventuali richieste. Nell'IMR ad ogni pin di richiesta (IR_n) è associato un bit attivando (o disattivando) il quale è possibile mascherare (o consentire) alcune richieste.

Il valore n relativo all'interrupt inviato alla cpu viene calcolato con la seguente formula:

$n = k + i$, dove i è compreso fra 0 e 7, k è una costante programmabile (questa operazione viene effettuata dal bios).

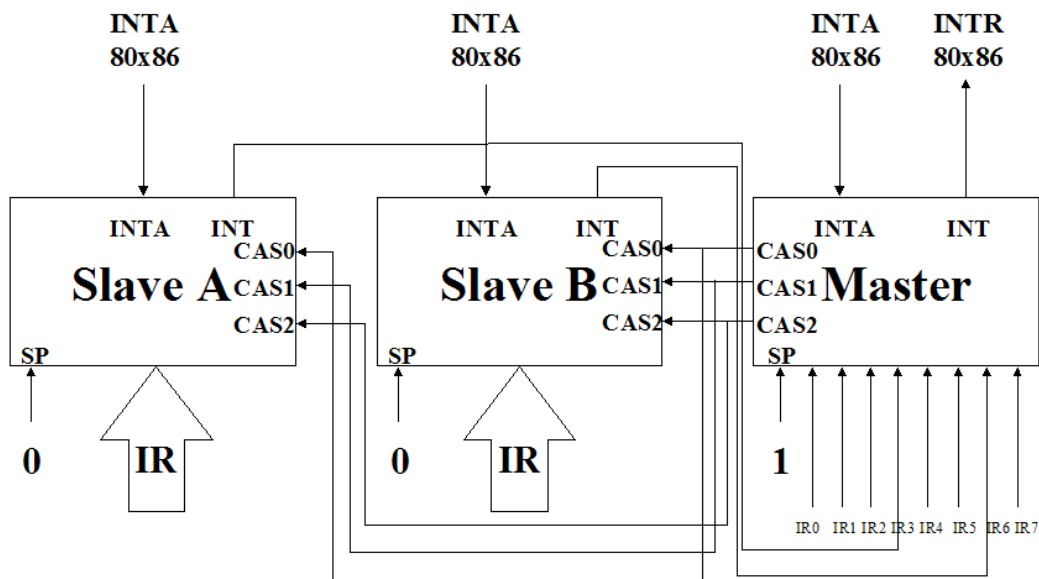
L'8259 può lavorare a priorità fissa (IR0 ha priorità massima fino a IR7 che ha priorità minima) oppure a priorità rotante. In quest'ultimo caso l'interrupt a priorità più alta non appena viene servito acquista priorità minima, si usa con dispositivi a priorità "paritetica" che devono essere serviti in modo democratico.

Servito un interrupt, prima di passare alla gestione di un nuovo interrupt è necessario il reset del controller pertanto, prima di eseguire la IRET le routine di gestione inviano una configurazione opportuna (chiamata EOI, End Of Interrupt) sugli ingressi del chip per resettare il sistema di gestione rendendolo idoneo a servire un nuovo interrupt

Interconnessione di più PIC

Per poter gestire più di 8 periferici è necessario interconnettere più PIC in cascata. E' possibile gestire fino ad un massimo di 64 livelli di interruzione: uno dei PIC funzionerà da master mentre tutti gli altri si comporteranno da slave.

Esempio



Prof. Marco Mezzalama 08/09

Il PIC master ha il compito di inviare il segnale di INT al processore nel caso in cui un dispositivo richiedesse un interrupt. Il pin di INT di ogni PIC slave è collegato ad uno degli otto pin (IR0-7) del PIC master. In questo caso la priorità degli IRn degli slave dipende dalla priorità dell' IRn del master a cui lo slave è collegato.

Inoltre è bene precisare che tutti i PIC(master e slave) ricevono il segnale di INTA proveniente dalla CPU. Ricevuto tale segnale risponderà solo il PIC che ne ha fatto richiesta, ponendo sul data bus il valore di n corrispondente.