L'ASSEMBLER dello Z80

di Pierluigi Panunzi

Interrupt

In questo numero ci occuperemo di un argomento molto interessante e stimolante, del quale il programmatore "medio", quello che lavora esclusivamente con linguaggi ad alto livello, quasi non si accorge oppure ne ignora l'esistenza: ci occuperemo in questa sede della gestione degli interrupt da parte dello Z80.

Dicevamo che si tratta di un argomento molto interessante: è senz'altro più complicato e delicato da gestire, praticamente solo a livello di linguaggio assemblativo, ed in alcuni casi risulta l'unico strumento a disposizione del programmatore per risolvere i problemi più complessi: di solito la gestione delle periferiche di I/O per mezzo di sofisticati circuiti integrati.

Analizziamo dunque il comportamento dello Z80 e le sue istruzioni nella gestione degli interrupt.

L'Interrupt Non Mascherabile (NMI)

Cominciamo con questo tipo, che è senz'altro il più importante: si tratta di un'interruzione che non si può in alcun modo disabilitare e per questo motivo è destinata alla gestione di apposite routine al verificarsi di eventi straordinari o particolarmente importanti.

Tale interrupt, nello Z80, prende il nome di NMI (Not Maskable Interrupt) e viene innescato quando nell'apposito piedino del componente arriva un impulso negativo.

Da questo istante lo Z80 completerà l'esecuzione dell'istruzione che stava eseguendo ed effettua, forzandola, una chiamata ad una subroutine: tutto va come se nel programma in corso di esecuzione lo Z80 incontrasse un'istruzione

```
CALL 0066H
```

in seguito alla quale viene eseguita la subroutine posta a partire dall'indirizzo 0066H.

La particolarità di tale subroutine è che il ritorno al programma interrotto non avviene con una RET, ma con la particolare istruzione

```
RETN
```

dove la "N" indica trattarsi del ritorno da una routine di gestione di un'interrupt non mascherabile.

Detto questo passiamo agli altri tipi di interrupt.

Gli Interrupt Mascherabili

Questo genere di interrupt viene innescato quando sul piedino INT dello Z80 perviene, da un dispositivo esterno, un impulso negativo.

Il nome "Mascherabili" in questo caso deriva dal fatto che, da programma, si possono disabilitare con l'istruzione

```
DI (Disable Interrupt)
```

rendendo lo Z80 "sordo" ad ogni interruzione (ma non all'NMI, come visto). Viceversa per riabilitare l'acquisizione di interrupt si ha a disposizione l'istruzione

```
EI (Enable Interrupt)
```

che riapre così la strada all'acquisizione di eventi esterni. Ora per quanto riguarda la gestione di questo tipo di interrupt, abbiamo tre possibili modi, a scelta del programmatore.

Tali modi, che prendono il nome di "Modo 0", "Modo 1", "Modo 2", sono selezionabili, rispettivamente, con una delle tre istruzioni

```
IM 0
IM 1
IM 2
```

Ognuna di queste modalità ha caratteristiche ben definite che si rispecchiano in tre funzionamenti completamente diversi: analizziamole in dettaglio, anticipando che in tutti e tre i casi il ritorno al programma interrotto avviene sempre tramite l'istruzione

```
RETI
```

che significa appunto "RETurn from Interrupt".

L'"Interrupt Mode 0"

Si tratta in particolare dell'unico "modo" disponibile sull'8080, dal quale lo Z80 ha preso le mosse: consiste nel salto ad una routine il cui indirizzo, in un certo senso, è fornito dal dispositivo esterno che ha generato l'interrupt.

In particolare tale dispositivo provvederà ad inviare sul DATA BUS un byte opportuno, rappresentante in generale il codice operativo di una particolare istruzione di salto, che prende il nome tecnico di "Restart" (RST).

Così, con un solo byte si può imporre al programma di saltare ad uno degli 8 punti prestabiliti della memoria: tali istruzioni di restart sono dunque otto e fanno saltare agli indirizzi indicati nella seguente tabella:

Istruzione	cod. oper.	salta a
RST 0H	0C7H	0000H
RST 8H	0CFH	0008H
RST 10H	0D7H	0010H
RST 18H	0DFH	0018H
RST 20H	0E7H	0020H
RST 28H	0EFH	0028H
RST 30H	0F7H	0030H
RST 38H	0FFH	0038H

Ora, dato che tali indirizzi distano l'uno dall'altro appena 8 byte, in genere in tali locazioni di memoria si inseriranno prevalentemente dei salti (JP nnnn) ad altri indirizzi dove la routine di gestione dell'interrupt potrà essere lunga quanto serve.

Il fatto che spaventa il programmatore alle prime armi è come si possa forzare tale istruzione sul DATA BUS. In questo caso ci vengono incontro i dispositivi stessi, ormai tutti "programmabili": è proprio in fase di programmazione del dispositivo (PIO, USART, CTC, DAC, ADC, ecc.), che viene inserito in un apposito registro tale byte che poi verrà automaticamente posto sul DATA BUS all'istante opportuno.

L'"Interrupt Mode 1" (Single line interrupt)

È questo il tipo di gestione più semplice, usato quando si ha un solo dispositivo esterno a generare interrupt e perciò si ha necessità di una sola routine di gestione: tale routine avrà come indirizzo iniziale 0038H, indirizzo forzato dallo Z80 con un'istruzione RST 38H, di codice FFH, generata perciò ponendo ad "1" tutti i bit del DATA BUS.

L'"Interrupt Mode 2" (Interruzioni vettorizzate)

Quest'ultima modalità è la più complessa, ma, come si vedrà, estremamente versatile e flessibile, consentendo la gestione di un numero molto grande di routine di interrupt, ciascuna associabile ad un certo dispositivo esterno.

Apriamo una parentesi per spiegare questo concetto: supponiamo che il nostro computer, oltre allo Z80, contenga vari dispositivi periferici, quali ad esempio una coppia di porte parallele ad 8 bit, un serializzatore-parallelizzatore di messaggi (USART) ed un Timer (in genere triplo).

Ora la coppia di porte parallele potrà generare ad esempio due tipi differenti di interrupt, relativi ad eventi che accadono sulle due porte; l'USART potrà invece generare una miriade di interrupt, ognuno relativo ad una condizione (messaggio pronto, errore di ricezione o di trasmissione, fine del messaggio, ecc). Infine il timer potrà generare tre interrupt in coincidenza con lo "scadere" dei tre clock stessi.

Ecco che perciò ogni interrupt avrà bisogno di una ben determinata routine di gestione, differente dalle altre.

Tornando perciò allo Z80, nel modo 2 abbiamo a disposizione una tabella di indirizzi di routine, ognuna relativa ad un certo interrupt, che prende il nome di "interrupt vector".

Questa tabella, in particolare, deve essere posta in memoria a partire da un indirizzo esadecimale terminante con 00: ad esempio 0300H, 1000H o 4E00H, dove appunto l'LSB è nullo.

In tale tabella si possono memorizzare gli indirizzi di 128 routine di interrupt, per un totale cioè di 256 byte.

Ora, in parole povere, il dispositivo che genera l'interrupt fornirà allo Z80 l'LSB di un indirizzo all'interno della tabella, indirizzo che a sua volta conterrà l'indirizzo reale a cui saltare per gestire l'interrupt.

Questo oscuro gioco di parole non è altro che un "indirizzamento indiretto" ottenuto, dicevamo, a partire dall'LSB fornito dal componente.

In sede di inizializzazione del programma, dopo aver stabilito il modo 2 con "IM 2", bisognerà comunicare allo Z80 l'MSB dell'"interrupt vector" (che potrà trovarsi perciò in un punto qualsiasi della memoria, ad un indirizzo che termina per 00): il valore dell'MSB in questione viene memorizzato nel registro "I" dello Z80, del quale finalmente conosciamo lo scopo: ne avevamo vagamente accennato nella prima puntata.

La memorizzazione si ottiene ponendo nell'accumulatore il valore dell'MSB desiderato e passandolo poi al registro I con l'istruzione

```
LD I,A
```

Analoga alla precedente è la "duale"

```
LD A,I
```

che consente eventualmente di verificare il valore dell'MSB contenuto nel registro I.

Ancora sulle "Restart"

Abbiamo parlato prima delle "Restart" come particolari configurazioni di bit che i dispositivi periferici pongono nel DATA BUS nel modo 0, per far sì che lo Z80 salti ad una di 8 routine.

A prescindere dal discorso degli interrupt, le "Restart" sono a tutti gli effetti delle istruzioni, che possono perciò trovarsi all'interno di un programma.

L'effetto di una "RST NNH" non è per nulla differente da una "CALL 00NNH", dal momento che in entrambi i casi si ha lo stesso tipo di esecuzione, con salvataggio nello stack dell'indirizzo di ritorno.

Il vantaggio dell'uso di una RST NNH è che, come detto, occupa un solo byte, contro i 3 di una CALL; invece lo svantaggio è che le subroutine indirizzate dalle RST possono essere solo 8 e devono essere collocate ad indirizzi ben prefissati.

In alcuni sistemi operativi (ad esempio il TRSDOS del TRS-80) a questi indirizzi sono poste routine di uso comune quali:
— input di un carattere da tastiera
— output su video di una stringa o di un carattere
— analisi del byte successivo, nella scansione del testo di un programma Basic
—output di un carattere sulla stampante.

Ancora una volta il vantaggio si ha nei programmi che richiamano parecchie volte tali routine: per ogni chiamata basta un solo byte invece di tre. MC