di Sergio Polini

I TSR e gli interrupt del BIOS

Approfitto del corsivo iniziale per scambiare due chiacchiere con un gentile lettore abruzzese, Cristian D'Aloisio di Liscia, in provincia di Chieti. Caro amico, il suo tentativo di intercettare la divisione per zero non funziona per due motivi. In primo luogo, lei prova a sostituire la routine associata all'INT 0 sperando di controllare così la divisione tra due numeri di tipo real, ma le routine del Turbo Pascal per i real non usano mai né DIV né IDIV. In secondo luogo, anche se si trattasse di due integer, non basta che la routine setti una variabile o un flag: occorre intervenire direttamente sui registri AX (che nel suo caso starebbe in [BP-4]) e IP (in [BP+2]); nel primo infatti va il risultato della divisione, mentre il secondo va incrementato (di due, nel Turbo Pascal 3) per evitare che, dopo l'INT 0, il controllo ritorni proprio alla istruzione che lo ha causato. L'«exception handling» è comunque un argomento di notevole interesse, al punto che se ne riparlerà più diffusamente nella rubrica. Le propongo quindi un prossimo appuntamento su queste pagine. A presto

La volta scorsa, dopo aver anticipato che un programma residente va attivato mediante un interrupt, abbiamo illustrato tutte le cautele che si devono prendere per evitare di interrompere in modo catastrofico le operazioni del DOS. Abbiamo cioè esaminato le aree di memoria interne del DOS, alcuni suoi flag e l'INT 28h, per precisare le condizioni che devono verificarsi perché possa essere attivato un TSR. Abbiamo anche visto come fare i conti con gli interrupt hardware, nonché una funzione *TSRAttivabile* che ritorna TRUE se a loro volta ritornano FALSE le funzioni *InDOS*, *InBIOS* e *In8259*; mentre quest'ultima viene compiutamente illustrata dal listato pubblicato il mese scorso, le altre due testano il valore di alcune variabili, per le quali ci rimane ancora da vedere come vengano gestite dalla unit. Queste variabili hanno tutte un nome formato da un «InInt» seguito da un numero, da intendere come numero di un interrupt. Cominceremo da quelli del BIOS, ovvero dagli interrupt 5 (Print Screen), 9 (tastiera), 10h (video) e 13h (dischi).

Interrupt 5

A gennaio abbiamo ricordato che, dichiarando **interrupt** una procedura, il compilatore produce codice che salva in entrata e poi ripristina in uscita tutti i registri. Comodo e scomodo allo stesso tempo. È comodo perché evita di dover procedere «manualmente» ad operazioni di questo tipo, come nelle prime versioni del Turbo Pascal si faceva con istruzioni **inline**; scomodo perché a volte si può desiderare una particolare manipolazione dei registri, che verrebbe vanificata dal finale ripristino dei valori salvati all'inizio. È possibile, in verità, fare in modo che al termine della procedura dichiarata **interrupt** alcuni registri abbiano valori scelti dal programmatore: si tratta, come più esaurientemente spiega il manuale, di dichiarare anche dei parametri con il nome dei registri che interessano — rispettando un certo ordine — e poi di lavorare su questi, ma le cose si complicano alquanto quando i valori da assegnare ai registri-parametro sono quelli che ad alcuni registri «veri»

```
var
  ...
  InInt5            : word;         (* > 0 se attivo INT 5 *)
  PrevInt5          : procedure;    (* INT 5 originario *)
  ...
procedure PUSHF; inline($9C);       (* mette i flag nello stack *)

procedure CLI; inline($FA);         (* disabilita gli interrupt *)

procedure NuovoInt5; interrupt;
begin
  Inc(InInt5);
  PUSHF;
  CLI;
  PrevInt5;
  Dec(InInt5)
end;
  ...
procedure Installa(Nome:string; Prog:Proc; ID:byte; Scan:byte; Shift:byte);
  ...
begin
  ...
  GetIntVec(5, Addr(PrevInt5));
  SetIntVec(5, Addr(NuovoInt5));
  ...
end;
begin
  ...
  InInt5        := 0;
  ...
end.
```

Figura 1 - La procedura NuovoInt5, che viene sostituita alla routine associata all'INT 5. Vengono mostrati anche altri frammenti della unit TSR.PAS: le dichiarazioni di InInt5 e PrevInt5 e le procedure inline PUSHF e CLI (viste già il mese scorso), nonché le due istruzioni della procedura Installa che salvano la routine originaria in PrevInt5 e le sostituiscono NuovoInt5, e la parte relativa a InInt5 della sezione di inizializzazione della unit.

sono stati assegnati da un interrupt: non c'è modo di leggere i valori dei registri «veri» se non facendo ricorso o a moduli in assembler o a istruzioni **inline**.

Distinguiamo quindi due tipi di interrupt: quelli che si limitano a «fare qualcosa» e quelli che «ritornano dei valori» in alcuni registri. L'interrupt 5 appartiene al primo gruppo, in quanto non fa altro che provocare l'invio alla stampante di una copia del contenuto del video. La figura 1 contiene il sorgente della procedura *NuovoInt5*, che viene associata all'INT 5 dalla procedura *Installa* (dato che si tratta della prima routine associata ad un interrupt che esaminiamo, la figura 1 richiama anche altri frammenti della unit già visti il mese scorso, e anticipa qualcosa di *Installa* e della sezione di inizializzazione della unit).

Vediamo quindi che si parte da un flag, *InInt5*, inizializzato a zero; la procedura *Installa* provvede a salvare in *PrevInt5*, variabile di tipo **procedure**, l'indirizzo della routine originaria, poi a sostituire a questa *NuovoInt5*, la quale incrementa il flag, simula l'attivazione dell'interrupt originario, quindi decrementa il flag.

L'effetto del tutto dovrebbe essere chiaro: si sostituisce all'INT 5 una routine che fa esattamente la stessa cosa, ma in più agisce su un flag che ci consente di sapere in ogni momento se quell'interrupt è in corso oppure no. In questo senso dicevamo il mese scorso che avremmo dovuto sostituire agli interrupt del BIOS altre routine che riproducessero il meccanismo del flag *InDOS*. Naturalmente non è sempre tutto così semplice.

```
procedure NuovoInt9; interrupt;
var
  Scan: byte;
begin
  Scan := Port[$60];
  PUSHF;
  CLI;
  PrevInt9;
  if (InInt9 = 0) and (not InTSRKey) then begin
    Inc(InInt9);
    STI;
    if ((Tasto = 0) or (Scan = Tasto)) and
     ((Mem[$0040:$0017] and $0F) = StatoShift) then
        InTSRKey := TRUE;
    Dec(InInt9)
  end
end;
```

Figura 2 - La routine che viene sostituita all'INT 9.

Interrupt 9

Anche l'INT 9 può considerarsi appartenente al gruppo degli interrupt che si limitano a «fare qualcosa». Questo «qualcosa» è in realtà alquanto complesso, in quanto consiste in tutta una serie di operazioni che permettono di passare dalle lettere minuscole a quelle maiuscole e viceversa, di attivare e disattivare il tastierino numerico, di usare l'INT 16h per leggere il carattere corrispondente al tasto premuto, e così via. Quello che però importa, appunto, è che un programma «legge» la tastiera mediante l'INT 16h, non mediante l'INT 9: questo scatta quando premiamo un qualsiasi tasto (è un interrupt hardware), quello quando lo decide un programma (è un interrupt software). È dall'INT 16h che desideriamo che ci vengano ritornati dei valori, non dall'INT 9, per il quale possiamo quindi procedere in modo analogo a quanto appena visto.

La differenza sta tutta nel fatto che alcuni tasti, o combinazioni di tasti, potrebbero proprio essere quelli che vogliamo attivino il nostro programma residente; alle azioni svolte dall'INT 9 dobbiamo quindi aggiungerne una: il riconoscimento dei tasti premuti; se questi sono quelli che vogliamo facciano partire il programma residente, viene assegnato TRUE alla variabile booleana *InTSRKey*. La combinazione di tasti designata viene passata in origine alla procedura *Installa* nei due parametri *Scan* e *Shift* (codice di scansione e stato dei tasti di shift), i cui valori vengono assegnati alle variabili *Tasto* e *StatoShift*. La procedura *NuovoInt9* provvede innanzitutto a leggere dalla porta 60h il codice di scansione del tasto premuto, poi chiama l'interrupt originario perché possa fare tutto quello che deve. A questo punto, se non si è già in *NuovoInt9* e se *InTSRKey* è FALSE, si incrementa il flag *InInt9* e si procede alla verifica. Se *Tasto* è uguale a zero (abbiamo visto a febbraio che si deve fare così se si vuole attivare il TSR con i soli tasti di shift) o è uguale al tasto prescelto, e se lo stato corrente dei tasti di shift (contenuto nei quattro bit «bassi» del byte all'indirizzo $0040:$0017) corrisponde anch'esso, si assegna TRUE a *InTSRKey* prima di decrementare *InInt9*.

Interrupt 10h e 13h

Ben diversa è la situazione con interrupt del secondo tipo, quelli che «ritornano dei valori». L'INT 10h, ad esempio, viene usato per impostare il modo del video (testo o grafica, monocromatico o a colori, ecc.), o forma e posizione del cursore; viene però anche usato per leggere questi stessi dati, o il carattere e l'attributo dove è il cursore, ecc. Le informazioni vengono ritornate in vari registri e, come già detto, un modulo in assembler o istruzioni **inline** sono l'unico modo per leggerli.

Nel caso poi dell'INT 13h, si devono anche preservare i flag. Questo interrupt svolge funzioni piuttosto delicate e importanti, quali resettare un disco, leggere lo stato di un disco dopo una precedente operazione, leggere, scrivere, verificare e formattare uno o più settori, ecc. Si tratta cioè delle routine

che più di altre consentono poi il regolare funzionamento del DOS, proprio in quanto «Disk Operating System»; è facile intuire quanta parte del DOS, pur offrendo una interfaccia più comoda e flessibile (per esempio: accesso ad un file mediante un *file pointer* invece che attraverso la conoscenza della sua collocazione fisica sul disco), ne faccia uso. È altrettanto facile capire quanto sia importante che eventuali condizioni d'errore segnalate dall'INT 13h vengano regolarmente trasmesse alle routine che lo usano. Il problema sta tutto qui, in quanto la situazione d'errore viene comunicata dall'INT 13h settando il flag *carry*.

Sappiamo che con la chiamata di un interrupt i flag vengono salvati nello stack e che vengono poi ripristinati dalla successiva istruzione IRET; se ci affidassimo a questo meccanismo, dopo l'esecuzione ritroveremmo i flag esattamente come erano prima, e quindi con una informazione assolutamente inattendibile. Dobbiamo quindi simulare un IRET «parziale», eliminando il POPF implicito e lasciando solo un RET 2, proprio come fa il BIOS.

Anche l'INT 13h, d'altra parte, usa i registri del microprocessore per trasmettere informazioni; si va da un semplice byte in AH a tutta una serie di valori in AX, BX, CX, DX, ES e DI, come nel caso di richiesta dei parametri del drive.

Per gestire adeguatamente tutti i diversi casi si potrebbero anche usare istruzioni **inline**, ma mi sembrerebbe inutilmente acrobatico. Si fa prima in assembler in quanto, non essendo necessario contrastare la manipolazione dei registri così come predisposta dal compilatore, basta limitarsi a riassegnare temporaneamente al registro DS l'indirizzo del *data segment* del programma residente. La figura 3 propone pertanto la prima parte del file TSRINT.ASM, con le due procedure da sostituire agli interrupt 10h e 13h e, per ognuna, la procedura che legge l'indirizzo della routine originaria e quella che associa la nostra all'interrupt. Vedremo in un altro numero la seconda parte del file, destinata alla sostituzione dell'INT 2Fh, la porta

```
; TSRINT.ASM
;=============================================================================
DATA            SEGMENT WORD    PUBLIC
                EXTRN   InInt10:WORD    ; dichiarate in TSR.PAS
                EXTRN   InInt13:WORD
                EXTRN   MultiplexID:BYTE
DATA            ENDS
;=============================================================================
CODE            SEGMENT BYTE    PUBLIC
                ASSUME  CS:CODE,DS:DATA
                PUBLIC  GetIntVec10, SetIntVec10
                PUBLIC  GetIntVec13, SetIntVec13
                PUBLIC  GetIntVec2F, SetInt2FVuoto, SetIntVec2F
;-----------------------------------------------------------------------------
PrevInt10       DD      ?               ; INT 10h originario
PrevInt13       DD      ?               ; INT 13h originario
PrevInt2F       DD      ?               ; INT 2Fh originario
;-----------------------------------------------------------------------------
; procedure NuovoInt10; interrupt;
NuovoInt10      PROC    FAR
                push    ds
                push    ax
                mov     ax,SEG DATA
                mov     ds,ax
                pop     ax
                inc     InInt10
                pushf
                cli
                call    cs:PrevInt10
                dec     InInt10
                pop     ds
                iret
NuovoInt10      ENDP
;-----------------------------------------------------------------------------
; procedure GetIntVec10; external;
GetIntVec10     PROC    NEAR
                mov     ax,3510h
                int     21h
                mov     word ptr cs:PrevInt10,bx
                mov     word ptr cs:PrevInt10+2,es
                ret
GetIntVec10     ENDP
;-----------------------------------------------------------------------------
; procedure SetIntVec10; external;
SetIntVec10     PROC    NEAR
                push    ds
                push    cs
                pop     ds
                mov     dx,OFFSET NuovoInt10
                mov     ax,2510h
                int     21h
                pop     ds
                ret
SetIntVec10     ENDP
;-----------------------------------------------------------------------------
; procedure NuovoInt13; interrupt;
NuovoInt13      PROC    FAR
                push    ds
                push    ax
                mov     ax,SEG DATA
                mov     ds,ax
                pop     ax
                inc     InInt13
                pushf
                cli
                call    cs:PrevInt13
                dec     InInt13
                pop     ds
                sti
                ret     2
NuovoInt13      ENDP
;-----------------------------------------------------------------------------
; procedure GetIntVec13; external;
GetIntVec13     PROC    NEAR
                mov     ax,3513h
                int     21h
                mov     word ptr cs:PrevInt13,bx
                mov     word ptr cs:PrevInt13+2,es
                ret
GetIntVec13     ENDP
;-----------------------------------------------------------------------------
; procedure SetIntVec13; external;
SetIntVec13     PROC    NEAR
                push    ds
                push    cs
                pop     ds
                mov     dx,OFFSET NuovoInt13
                mov     ax,2513h
                int     21h
                pop     ds
                ret
SetIntVec13     ENDP
;-----------------------------------------------------------------------------
```

Figura 3 - La prima parte del file TSRINT.ASM, con le procedure relative alla intercettazione degli interrupt 10h e 13h. Il file può essere assemblato sia con il MASM della Microsoft che con il TASM della Borland. Le procedure relative all'INT 2Fh verranno mostrate in un prossimo numero.

```
procedure Beep;
begin
  Sound(100);
  Delay(50);
  NoSound
end;

procedure NuovoInt8; interrupt;
begin
  PUSHF;
  CLI;
  PrevInt8;
  if InInt8 = 0 then begin
    Inc(InInt8);
    STI;
    if InTSRKey and (not InTSRProg) then begin
      if TSRAttivabile then begin
        if Test8087 > 0 then FSAVE;
        InTSRProg := TRUE;
        EseguiTSR;
        InTSRProg := FALSE;
        if Test8087 > 0 then FRSTOR
      end
      else begin
        InTSRKey := FALSE;
        Beep
      end
    end;
    Dec(InInt8)
  end
end;
```

Figura 4 - La procedura che viene associata all'INT 8. Viene mostrata anche la procedura Beep, che emette un (discreto) segnale sonoro se, pur avendo l'utente premuto la combinazione di tasti che attiverebbe il TSR, l'attivazione non è possibile.

d'accesso «ufficiale» ai programmi residenti.

Interrupt 8

L'INT 8 scatta automaticamente circa 18,2 volte al secondo. Si tratta di un interrupt hardware che ha tre funzioni: tenere il conto dei suoi scatti dal momento dell'accensione della macchina (in modo che l'INT 1Ah ci possa poi dire che ore sono), spegnere il motore dei floppy disk dopo qualche secondo di inattività, chiamare l'INT 1Ch, al quale l'utente può associare una propria routine.

Intercettare l'interrupt 8 non ci è utile tanto per evitare di attivare il TSR nel momento sbagliato, ma, al contrario, proprio per attivarlo. La routine associata all'INT 9, come abbiamo visto, assegna TRUE alla variabile *InTSRKey* quando l'utente preme la combinazione dei tasti prescelta per l'attivazione. Poiché l'INT 8 scatta ogni 55 millisecondi, consente un controllo praticamente costante sul valore di questa variabile. Se la trova TRUE, se non si è già nell'esecuzione del TSR (variabile *InTSRProg*), se infine DOS e BIOS possono essere interrotti, viene chiamata la procedura *EseguiTSR*, la quale a sua volta provvede (come vedremo nel prossimo numero) a lanciare il programma residente.

Un'ulteriore complicazione può essere rappresentata dalla presenza di un coprocessore numerico, segnalata da un valore diverso da zero della variabile predefinita *Test8087*. L'80x87 opera mediante un proprio stack interno a otto registri; i risultati dei calcoli vengono sempre depositati in questo stack, e si richiedono separate istruzioni per muovere i valori da qui alla memoria. Se un TSR venisse attivato dopo il calcolo ma prima del trasferimento del risultato dal coprocessore alla memoria, e se il TSR a sua volta usasse il coprocessore, il programma interrotto potrebbe sembrare... dare i numeri. Ecco perché si usano le procedure **inline** FSAVE e FRSTOR, che salvano e poi rispristinano lo stato dell'80x87.

Interrupt 28h

Questo non è in verità un interrupt del BIOS, ma del DOS: ricorderete infatti che si tratta di quell'interrupt chiamato ripetutamente dalle funzioni 01h-0Ch del DOS quando queste rimangono in attesa di un evento quale la pressione di un tasto.

Lo scopo è quello di consentire l'esecuzione di altre attività mentre il DOS è praticamente fermo. Normalmente all'INT 28h è associata una semplice istruzione IRET, ma nulla vieta di agganciarvi una routine che, se se ne verificano tutte le condizioni, faccia partire il programma residente. La figura 5 mostra appunto questa routine, molto simile a quella della figura 4, associata all'INT 8.

Con ciò abbiamo terminato l'esame delle condizioni e dei meccanismi di attivazione di un programma residente. Il mese prossimo vedremo cosa succede (meglio: cosa deve succedere) quando il TSR assume il controllo delle operazioni.

```
procedure NuovoInt28; interrupt;
begin
  PUSHF;
  CLI;
  PrevInt28;
  if InInt28 = 0 then begin
    Inc(InInt28);
    if InTSRKey and (not InTSRProg) then begin
      if TSRAttivabile then begin
        if Test8087 > 0 then FSAVE;
        InTSRProg := TRUE;
        EseguiTSR;
        InTSRProg := FALSE;
        if Test8087 > 0 then FRSTOR
      end
      else
        InTSRKey := FALSE
    end;
    Dec(InInt28)
  end
end;
```

Figura 5
La procedura associata all'INT 28h.