Orologio Digitale

Obiettivo

Il nostro intento è creare un programma ed implementarlo in linguaggio Assembly 8086, che stampi a schermo un orologio digitale. Inizialmente il valore sarà sincronizzato con l'orologio di sistema, in seguito la modifica avverrà attraverso la gestione del TIC.

Analisi Hardware

Il TIC

Il TIC, il temporizzatore di sistema, è un dispositivo diviso in 4 canali. Ogni canale è composto da un contatore a 16 bit che si decrementa con una frequenza

$$f = 1.19318 MHz$$

Ogni volta che il contatore raggiunge il valore zero, viene generato un interrupt hardware sul piedino INTR del microprocessore (interrupt mascherabile), dopodichè il conteggio riparte.
Il costruttore assicura che il TIC genera un interrupt di priorità massima rispetto a tutti gli altri.
Utilizzando il valore della frequenza fornita dal costruttore, si ricava che, se il contatore parte dal valore massimo (FFFFh), esso si azzera ogni

$$N = 18.2 \frac{volte}{\sec ondo}$$

Il TIC ha compiti molto importanti all'interno del sistema microprocessore: tra le attività che svolge, c'è quella fondamentale di eseguire il refresh della RAM dinamica.
Uno dei canali del TIC è reso disponibile per il programmatore: ogni volta che il rispettivo contatore si azzera, si genera un interrupt hardware al vettore 8.
Tuttavia, la funzione corrispondente è importante per il sistema operativo, quindi non possiamo semplicemente sovrascriverla cambiando il valore del vettore 8. Per nostra fortuna questa situazione era stata prevista dai costruttori, che hanno fatto in modo che, alla fine della routine puntata dal vettore 8, sia generato un secondo interrupt al vettore 1Ch. E questo interrupt è generato apposta per il programmatore, quindi possiamo tranquillamente far eseguire una nostra routine.

Analisi Software

Programmazione della tabella dei vettori

La tabella dei vettori non è scrivibile direttamente dal programmatore, ma solo attraverso l'utilizzo di apposite routine fornite dal sistema operativo sotto forma di servizi interrupt software.
In particolare, il DOS (int `21h`) fornisce il servizio `25h`:

- Servizio `25H`, set interrupt vector (in AL=interrupt number, DS:DX=new vector)

In pratica, le azioni da eseguire per settare una nuova routine nella tabella dei vettori sono

```
Blocco Interrupt
AL ← Numero del vettore
DS ← Code Segment
DX ← Indirizzo della routine
Riattivazione interrupt
!!! Ripristino del DS !!!
```

Va notato che, per ricavare l'indirizzo della routine, è sufficiente riferirsi ad essa con la sua *label*, il nome che le abbiamo assegnato: infatti la label non è altro che un modo per definire un'etichetta mnemonica al posto di un indirizzo: le label nell'area codice definiscono un indirizzo della zona codice (quindi con segmento CS), pertanto il nome di una routine corrisponde al punto di entrata della stessa.
Va precisato che la manipolazione della tabella degli interrupt deve avvenire sotto protezione dalla rilevazione di interrupt, per evitare problemi causati dalla modifica parziale dei registri o dall'aggiornamento della tabella stessa.

Gestione del tempo con il TIC

Abbiamo detto che il TIC genera un interrupt hardware al vettore `8` (e, conseguentemente, all'`1Ch`) 18.2 volte al secondo. Quindi la nostra routine viene eseguita appunto questo numero di volte ogni secondo. Se vogliamo rilevare il passaggio di un secondo, dobbiamo eseguire ciò che vogliamo solo ogni 18 chiamate alla routine.
Ma manca una questione: ogni 5 secondi è necessario aspettare un'esecuzione in più (19 in tutto) in modo da "pareggiare" gli scompensi che si sono creati eseguendo il codice ogni 18 volte (invece che ogni 18.2).
Infatti abbiamo che

```
18.2 * 5 = 91 = 4*18 + 19
```

Va precisato subito che, all'entrata della routine di servizio, il data segment è sporco (viene sporcato dalla routine puntata dal vettore 8),

quindi deve essere ripristinato con un'opportuna istruzione. In realtà sarà sufficiente spostare nel registro DS il valore @data, che contiene automaticamente il valore iniziale del DS impostato all'avvio del programma.

```
DS ← @data
```

Stampa dell'orologio

Possiamo immaginare l'orologio come tre contatori (secondi, minuti, ore) che si incrementano automaticamente ogni secondo (il passaggio del tempo si può rilevare con il TIC). Se la nostra routine lavora con dei dati già presenti in memoria (e quindi i contatori sono visti come delle label che puntano a dei byte), sarà quindi sufficiente inizializzare quest'area al valore dell'orario attuale per ottenere l'orologio.

Notare che la stampa deve avvenire comunque attraverso simboli ASCII, quindi abbiamo due strade da percorrere:

1. Lavoriamo direttamente con contatori in formato ASCII
2. Lavoriamo con contatori numerici ma prima della stampa convertiamo il tutto in ASCII

La via che ho scelto, perché mi sembrava la più comoda, è quella di gestire i contatori in formato ASCII. In pratica, necessitiamo di 6 byte:

1. Decine delle ore ('0'…'2')
2. Unità delle ore ('0'…'9')
3. Decine dei minuti ('0'…'5')
4. Unità dei minuti ('0'…'9')
5. Decide dei secondi ('0'…'5')
6. Unità dei secondi ('0'…'9')

La nostra routine di servizio dovrà quindi aggiornare le unità dei secondi e, se necessario, aggiornare in cascata tutti gli altri contatori coinvolti (sarà sufficiente controllare il superamento del valore massimo).

```
Rilevo se è passato un secondo
     Aggiorno il contatore unità secondi
          Se è arrivato a '9' + 1
               Aggiorno le decine dei secondi
                    Se... e così via fino alle unità delle ore
     Eseguo la stampa
```

Discorso a parte meritano le ore, perché, oltre a gestire il riporto unità – decine, è necessario prevedere un controllo quando l'orario arriva a 24: in questo caso, infatti, tutto il dispositivo deve essere azzerato.

```
Se l'orario è 24
     ore ← '0'
     ore+1 ← '0'
```

In realtà è sufficiente azzerare la parte delle ore, perché le altre parti, per come è strutturato il nostro algoritmo, sono già state azzerate in cascata prima di poter arrivare alla modifica dei contatori dell'ora.

Va notato che le routine di gestione degli interrupt non sono mai generiche: anche nel nostro caso esse si basano su dati presenti in memoria a particolari label, quindi non saranno adattabili ad altri casi.

L'ultima cosa da fare è sistemare le locazioni di memoria utilizzate per la routine di servizio: per farlo è necessario rilevare l'orario di sistema. Ho scelto di utilizzare l'interrupt software 1Ah che permette di avere l'orario in formato BCD compattato, facilmente scomponibile in ASCII con una routine di mascheratura.

- Servizio 2, Read real time clock (returns in BCD : CH=hours, CL=minutes, DH= seconds)

Una volta sistemata l'ora di partenza, andrà solamente sistemata la tabella degli interrupt e il nostro problema è risolto.
Una precisazione: se non si esegue una prima stampa nel main, l'orario verrà stampato per la prima volta quando la routine di gestione viene effettivamente eseguita completamente, ossia dopo il primo secondo di esecuzione. Per ovviare il problema, come detto, è sufficiente stampare l'orario anche all'inizio del programma.

```
Rileva l'ora di sistema
Stampa l'ora di sistema
Imposta la routine di gestione degli Interrupt
Rimani in esecuzione indeterminatamente
```

Il programma rimane in loop infinito, in modo che l'ora sia stampata continuamente finché l'utente non interrompe l'esecuzione manualmente.

Jump "lontani"

Per come sono interpretate le istruzioni di jump condizionale, esse non permettono un salto di oltre 256 byte. Se ciò dovesse essere necessario, come nel mio caso, si ricorre ad un piccolo trucchetto, che consiste nel far saltare il programma ad un punto "morto", ossia dove l'esecuzione normalmente non arriverebbe, e porre in quel punto un JMP secco alla label che ci interessava.

Va notato che i punti morti del nostro programma sono essenzialmente dopo tutte le istruzioni JMP non condizionali: infatti, arrivati a quel punto, l'esecuzione si sposterà in un'altra zona, impedendo l'esecuzione della successiva istruzione.

Struttura del programma

Programma principale

1. Inizializza le variabili necessarie per il programma
2. Rileva l'ora di sistema
3. Effettua la prima stampa
4. Imposta la tabella degli interrupt

Routine "orol"

1. Ripristina il data segment
2. Routine associata all'interrupt generato dal TIC
3. Rileva il passaggio di un secondo
4. Aggiorna tutti i rispettivi contatori
5. Stampa l'orario
6. Richiede le locazioni di memoria
   - conta: inizializzato a 18, indica il numero di chiamate a vuoto prima che sia passato un secondo
   - pareggio: inizializzato a 5, indica ogni quanti secondi deve essere eseguita una chiamata a vuoto per pareggiare il tempo
   - Locazioni di memoria per ore, minuti e secondi (2 byte ognuna); N.B. Il byte a indirizzo basso di ogni coppia indica le decine. Inoltre la struttura che devono avere questi byte è quella della stampa, quindi ORE, separatore, MINUTI, separatore, SECONDI, carattre $ come terminatore della sequenza
7. Sistema i contatori per la rilevazione successiva

Routine "stampaora"

1. La routine riceve in ingresso una coppia di byte che indica la posizione in cui stampare e la stringa dell'orario (formato ore, separatore, minuti, separatore, secondi, separatore, carattere $ come terminatore della convenzione DOS). La stringa termina con $ perché per la stampa si utilizza l'interrupt 21h del DOS.
2. La routine si posiziona sullo schermo e stampa la stringa

Routine "scompatta"

1. La routine riceve in ingresso un dato BCD compattato a 8 bit e lo scompatta nelle sue due componenti (decine e unità), depositando i risultati in due byte puntati dal registro SI ([SI] e [SI+1])

Costanti di servizio

Costante "separa"
E' un byte ASCII che indica il carattere da stampare per separare ore, minuti e secondi: generalmente è ':'

Costanti "maschera_alta" e "maschera_bassa"
Sono le due maschere utilizzate dalla routine *scompatta* per prelevare la parte alta e la parte bassa del numero BCD compattato.

Costante "posizione_stampa"
Indica la posizione utilizzata dall'interrupt 10h del video in cui posizionare il cursore per la stampa. E' formata da due byte: il primo indica la riga, il secondo la colonna

Costanti "num_pareggio" e "tic_secondo"
Costanti utilizzate per la gestione del TIC. La prima indica ogni quanti secondi deve essere eseguita una chiamata "a vuoto" della routine di gestione dell'interrupt per pareggiare il tempo. La seconda indica quante chiamate della routine devono essere effettuate prima che sia passato un secondo effettivo.

Codice

```
;--------------------------------------------------------------------------------
; STUDENTE:            Andrea Asta
; PROGRAMMA:           orologio.asm
; DESCRIZIONE: Stampa un orologio digitale sfruttando gli interrupt del TIC
;--------------------------------------------------------------------------------

; Segmentazione del DOS
        DOSSEG
; Modello di memoria
        .MODEL  small
; Memoria per lo stack
        .STACK  100h

;--------------------------------------------------------------------------------
; Data segment
;--------------------------------------------------------------------------------
        .DATA

; Cose di servizio
separa                  EQU     ':'
maschera_alta           EQU     11110000b
maschera_bassa          EQU     00001111b
posizione_stampa        EQU     0520h

; Propri dati
ore             db      ? , ?
                db      separa
minuti          db    ? , ?
                db      separa
secondi         db    ? , ?
                db    '$'
```

```
; Richieste per il funzionamento del tic
num_pareggio EQU    5
tic_secondo  EQU    18
conta        db     ?
pareggio     db     ?


;--------------------------------------------------------------------------------
; Code Segment
;--------------------------------------------------------------------------------
        .CODE

stampaora       PROC
        ; Stampa la stringa HH-MM-SS
        ; DX -> Posizione di stampa
        ; DS:SI -> Stringa completa HH-MM-SS

        PUSHF
        PUSH    AX BX DX

        ; Riposiziono il cursore
        MOV     BH , 0
        MOV     AH , 2
        INT     10h

        ; Stampo la stringa
        MOV     DX , SI
        MOV     AH , 9
        INT     21h

        POP     DX BX AX
        POPF
        RET
        ENDP

scompatta       PROC

        PUSHF
        PUSH    AX

; Riceve in ingresso un dato 8 bit (AL) BCD compattato
; e un puntatore ad una zona di memoria di 2 byte (SI)
; Mette in SI <- decine e in SI + 1 <- Unità

        ; Prelievo le decine
        MOV     AH , AL
        AND     AH , maschera_alta
        SHR     AH , 4
        ADD     AH , '0'
        MOV     BYTE PTR [SI] , AH

        ; Prelievo le unità
        MOV     AH , AL
        AND     AH , maschera_bassa
        ADD     AH , '0'
        MOV     BYTE PTR [SI+1] , AH

        POP     AX
        POPF
        RET
        ENDP

orol    PROC
```

```
        PUSH    AX DX DS ES SI BX

        ; Ripristino DS
        MOV     AX , @data
        MOV     DS , AX

        ; E' passato un secondo?
        CMP     [conta] , 0
        JNZ     lamp_supp

        ; E' stato anche effettuato l'eventuale "pareggio"?
        CMP     [pareggio]   ,  0
        JZ      lamp_supp2

             ; E' passato il mio secondo effettivo
             ;-----------------------------------------
             ; Scrivi ciò che vuoi fare nella routine

             ; Incremento i secondi (le unità)
             INC      BYTE PTR [secondi+1]
             CMP      BYTE PTR [secondi+1] , '9' + 1
             JNZ      lamp_x5
             MOV      BYTE PTR [secondi+1] , '0'

             ; E in cascata eventualmente tutto il resto!
             INC      BYTE PTR [secondi]
             CMP      BYTE PTR [secondi] , '5' + 1
             JNZ      lamp_x5
             MOV      BYTE PTR [secondi] , '0'

             INC      BYTE PTR [minuti+1]
             CMP      BYTE PTR [minuti+1] , '9' + 1
             JNZ      lamp_x5
             MOV      BYTE PTR [minuti+1] , '0'

             INC      BYTE PTR [minuti]
             CMP      BYTE PTR [minuti] , '5' + 1
             JNZ      lamp_x5
             MOV      BYTE PTR [minuti] , '0'

             INC      BYTE PTR [ore+1]
             CMP      BYTE PTR [ore+1] , '9' + 1
             JNZ      lamp_x5check
             MOV      BYTE PTR [ore+1] , '0'

             INC      BYTE PTR [ore]
             JMP      lamp_x5

             ; Label di supporto per i far JMP
lamp_supp:   JMP      lamp_x6
lamp_supp2:  JMP      lamp_x3

lamp_x5check:; Devo controllare che non siano le 24!
             CMP      BYTE PTR [ore+1], '4'
             JNZ      lamp_x5
             CMP      BYTE PTR [ore], '2'
             JNZ      lamp_x5
             MOV      [ore] , '0'
             MOV      [ore+1] , '0'

lamp_x5:        ; Stampa
                MOV         DX , posizione_stampa
                LEA         SI , ore
```

```
                CALL        stampaora

        ;-----------------------------------------

              ; Aggiorno i vari contatori del tempo
              MOV     [conta] , tic_secondo
              DEC     [pareggio]
              JMP     lamp_x2

lamp_x6:      JMP     lamp_x2

lamp_x3:      MOV     [pareggio]   ,  num_pareggio
              JMP     lamp_x4

lamp_x2:      DEC     [conta]

lamp_x4:      POP     BX SI ES DS DX AX
              IRET
              ENDP

; Inizio del programma
Begin:        StartupCode

                MOV     [conta] , tic_secondo
                MOV     [pareggio] , num_pareggio

                ; Prelevo l'orario
                MOV     AH , 2
                INT     1Ah

                ; Scompatto i dati
                MOV     AL , CH
                LEA     SI , ore
                CALL    scompatta

                MOV     AL , CL
                LEA     SI , minuti
                CALL    scompatta

                MOV     AL , DH
                LEA     SI , secondi
                CALL    scompatta

                ; Prima stampa
                MOV     DX , posizione_stampa
                LEA     SI , ore
                CALL    stampaora


        ; Aggiorno la tabella degli interrupt, ma prima mi proteggo da altri INT
        CLI

        ; Salvo DS
        PUSH    DS

        ; Sistemo i valori da impostare nella tabella
        MOV     AX , CS
        MOV     DS , AX
        LEA     DX , orol

        ; Imposto la tabella con la mia routine (orol)
        MOV     AH , 25h
        MOV     AL , 1Ch
```

```
INT     21h

; E ora ripristino tutto e tolgo la protezione
POP     DS
STI

; Rimango in attesa
JMP     $

END             Begin
```