Esercizi Assembly 10

M. Rebaudengo – R. Ferrero

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1

- Si scriva un programma in grado di leggere periodicamente le porte A e B del modulo 8255 e fornisca sulla porta C la somma dei due valori (da considerare espressi in complemento a 2). Eventuali condizioni di *overflow* devono essere segnalate con il valore 0FFh.
- La lettura deve essere fatta a intervalli regolari di 5 ms: si realizzi un opportuno ciclo di ritardo, considerando una frequenza di *clock* di 10 MHz e una media di 10 cicli di clock per istruzione.

Codice

```
PORTA       EQU 80h
PORTB       EQU PORTA+1
PORTC       EQU PORTA+2
CONTROL     EQU PORTA+3
DELAY       EQU 5000
; cicli di attesa: 0.005 s * 10MHz / (10 istr/clk)

#START=8255.exe#

            .model small
            .stack
            .data
            .code
            .startup

            MOV DX, CONTROL
            MOV AL, 10010010b
            OUT DX, AL
ciclo:      MOV DX, PORTA
            IN AL, DX
            MOV AH, AL
            MOV DX, PORTB
            IN AL, DX
            MOV DX, PORTC
            ADD AL,AH
            JNO next
            MOV AL, 0FFh
next:       OUT DX, AL
            MOV CX, DELAY
attesa:     LOOP attesa

            JMP ciclo ; ciclo infinito
            .exit

            end
```

Esercizio 2

- Si scriva una procedura in grado di leggere un intero (compreso tra 0 e 7) dalla porta A del modulo 8255, e inverta il valore logico del bit della porta C che ha come indice tale intero.
- Per la lettura del contenuto della porta C programmata in modo 0 – output si veda Q16 all'indirizzo

https://web.archive.org/web/20131006011609/http://www.intel.com/design/archives/periphrl/docs/7190.HTM

Codice

```
PORTA   EQU 80h
PORTB   EQU PORTA+1
PORTC   EQU PORTA+2
CONTROL EQU PORTA+3
#START=8255.exe#

        .model small
        .stack
        .data
        .code
        .startup

        MOV DX, CONTROL
        MOV AL, 10010000b
        OUT DX, AL

        MOV DX, PORTC
        MOV AL, 055h  ; valore di prova
        OUT DX, AL

        CALL rdwr

        .exit
```

```
rdwr    PROC
        MOV DX, PORTA
        IN AL, DX
        MOV CL, AL
        MOV BL, 1
        SHL BL, CL

        MOV DX, PORTC
        IN AL, DX
        XOR AL, BL
        OUT DX, AL

        RET
rdwr    ENDP

        end
```

Esercizio 3

- Sia dato un sistema basato su processore 8086, con il gruppo A del modulo 8255 configurato in modo 0 e la porta A in input
- Si scriva un programma che, periodicamente, lanci una procedura in grado di
  - Leggere dalla porta A dell'8255 un byte proveniente da una periferica esterna
  - Accorpare i byte ricevuti in due chiamate a funzione successive in una *word* corrispondente a un valore intero (è ricevuto prima il *byte* più significativo);
  - Inserire le *word* acquisite in due vettori, `vet_pari` e `vet_disp`, a seconda che ciascuna di esse sia, rispettivamente, pari o dispari.

Esercizio 3 [cont.]

- Si assuma di avere definito e inizializzato le seguenti strutture:

```
vet_pari dw DIM DUP (?)
vet_disp dw DIM DUP (?)
ind_pari db 0
ind_disp db 0
```

  dove DIM è una costante di valore massimo 255, mentre `ind_pari` e `ind_disp` contengono l'indice dove scrivere il valore successivo per ciascuno dei due vettori. Qualora uno dei vettori risultasse pieno, la memorizzazione deve ripartire dalla prima posizione (come in un *buffer* circolare).
- Esempio:

  Sequenza di byte ricevuti: `01, 0A, 31, 28, 33, 45`
  vet_pari: `010A, 3128`
  vet_dispari: `3345`

Codice

```
DIM         EQU 3
PORTA       EQU 80h
CONTROL     EQU PORTA+3
            #start=8255.exe#
            .model small
            .data
vet_pari    dw DIM DUP (?)
vet_disp    dw DIM DUP (?)
ind_pari    db 0
ind_disp    db 0
primo_flag  db 0
temp        db ?
            .stack
            .code

            .startup
            MOV DX, CONTROL
            MOV AL, 10010000b ; Gruppo A: modo 0, porta A input
            OUT DX, AL

ciclo:      CALL read_data
            JMP ciclo ; ciclo infinito!
            .exit
```

Codice [cont.]

```
read_data   PROC
            PUSH AX
            PUSH BX
            PUSH DX

            MOV DX, PORTA
            IN AL, DX
            TEST primo_flag, 1
            JZ asp_pross

            XOR BH, BH
            MOV AH, temp
            TEST AL, 1
            JZ pari

dispari:    CMP ind_disp, DIM
            JNE next_disp
            MOV ind_disp, 0
next_disp:  MOV BL, ind_disp
            SHL BX, 1
            MOV vet_disp[BX], AX
            INC ind_disp
            MOV primo_flag, 0
            JMP ritorno

pari:       CMP ind_pari, DIM
            JNE next_pari
            MOV ind_pari, 0
next_pari:  MOV BL, ind_pari
            SHL BX, 1
            MOV vet_pari[BX], AX
            INC ind_pari
            MOV primo_flag, 0
            JMP ritorno

asp_pross:  MOV temp, AL
            MOV primo_flag, 1

ritorno:    POP DX
            POP BX
            POP AX
            RET
read_data   ENDP

            end
```

Esercizio 4

- Siano dati:
  - un vettore di *byte* `vet1` contenente DIM elementi (DIM dichiarato come costante)
  - un vettore di *word* `vet2`, della stessa dimensione, non inizializzato.
- Si scriva una procedura **`extract`** in linguaggio Assembly 8086 in grado di ottenere, a partire dai dati espressi come *byte* in vet1, una sequenza di valori su 12 bit componendo *nibble* (insieme di 4 bit) di dati consecutivi, come esemplificato di seguito:

Esercizio 4 [cont.]

dati	risultati
0010 1101	
0100 0010	0000 **0010 1101 0100**
0100 1011	0000 0010 0100 1011
1000 0001	
0110 0011	0000 1000 0001 0110
1100 0000	0000 0011 1100 0000
1111 1111	
0000 1011	0000 1111 1111 0000

- La procedura deve memorizzare i risultati ottenuti in `vet2` (azzerando il *nibble* più significativo), procedendo fino a quando sono disponibili dati su `vet1` sufficienti a comporre un risultato; deve inoltre restituire al programma chiamante il numero di risultati memorizzati attraverso il registro DI.
- Si supponga che il programma chiamante lanci la procedura una volta sola. Si utilizzino variabili globali per l'indirizzamento dei vettori.

Codice

```
DIM     EQU 11
        .model small
        .stack

        .data
vet1    db 45, 66, 74, 129, 99,
        192, 255, 11, 98, 230, 187
vet2    dw DIM DUP(?)
num     db 9

        .code
        .startup
        call extract
        .exit

extract proc
        xor SI, SI
        xor DI, DI
        mov CL, 4
        mov CH, 0
ciclo:  mov AH, vet1[SI]
        inc SI
        cmp SI, DIM
        je  fine
        mov AL, vet1[SI]
        mov BH, AL
        shr AX, CL
        mov vet2[DI], AX
        add DI, 2
        inc SI
        cmp SI, DIM
        je  fine
        mov BL, vet1[SI]
        and BX, 0fffh
        mov vet2[DI], BX
        inc SI
        add DI, 2
        cmp SI, DIM
        jnz ciclo

fine:   shr DI, 1
        ret
extract endp

        end
```