Esercizi Assembly 5

M. Sonza Reorda – M. Grosso

Politecnico di Torino
Dipartimento di Automatica e Informatica

Esercizio 1

- Scrivere un programma in Assembly 8086 che, dati due operandi `opa` e `opb` di tipo *word* in memoria, del valore rispettivo di 2043 e 5, esegua un'operazione tra interi scelta dall'utente e salvi il risultato nella variabile *word* `res`
- A seconda del carattere digitato dall'utente, il programma deve eseguire:
  - 1 → `res = a+b`
  - 2 → `res = a-b`
  - 3 → `res = a*b`
  - 4 → `res = a/b` (divisione intera).

Implentazione

- Occorre implementare un costrutto *switch*:

```
switch (espressione)
    {
    case val1:      sequenza1;
            break;
    case val2:      sequenza2;
            break;
    ...
    default:        sequenza_def;
    }
```

- Si possono utilizzare:
  - operazioni di *compare* e salti condizionati a blocchi di istruzioni
  - una tabella di *jump* e un'unica istruzione di salto incondizionato.

Codice

```
.model small
.stack
.data
opa dw 2043
opb dw 5
res dw ?

.code
.startup
mov ah, 01h
int 21h
cmp al, '1'               ; implementazione di costrutto 'switch'
je  somma
cmp al, '2'
je  differenza
cmp al, '3'
je  moltiplicazione
cmp al, '4'
je  divisione
jmp fine                 ; ramo di default
```

Codice [cont.]

```
somma:     mov ax, opa
           add ax, opb
           mov res, ax
           jmp fine
differenza: mov ax, opa
           sub ax, opb
           mov res, ax
           jmp fine
moltiplicazione: mov ax, opa
           mul opb
           mov res, ax
           jmp fine
Divisione: mov ax, opa
           xor dx, dx   ; azzeramento registro dx
           div opb
           mov res, ax
fine:
.exit
end
```

Codice (alternativa)

```
.model small
.stack
.data

opa dw 2043
opb dw 5
res dw ?

tab dw somma, differenza, moltiplicazione, divisione   ; tabella di offset di codice

.code
.startup

mov ah, 01h
int 21h
sub al, '0'
dec al
mov ah, 0
mov bx, ax
shl bx, 1   ; moltiplicazione per 2
```

Codice (alternativa) [cont.]

```
jmp tab[bx]

somma:     mov ax, opa
           add ax, opb
           jmp fine
differenza: mov ax, opa
           sub ax, opb
           jmp fine
moltiplicazione: mov ax, opa
           mul opb
           jmp fine
divisione: mov ax, opa
           xor dx, dx   ; azzeramento registro dx
           div opb

fine:
mov res, ax
.exit
end
```

Esercizio 2

- Scrivere un programma che acquisisca due variabili di tipo *byte* opA e opB in formato binario da tastiera (sequenza di ‘0’ e ‘1’ digitati dall’utente) e scriva il risultato di una operazione logica *bitwise* in una variabile di tipo *byte* `ris`. L’espressione logica bit-a-bit in questione è
- C = NOT(A AND (NOT(B))) OR (A XOR B).

Implementazione

- Acquisizione di valori binari
  - Necessaria conversione di caratteri ASCII letti da tastiera
  - L'utente introduce valori binari (es.: "00011100") separati da ENTER
- Operazioni logiche su *byte* effettuate con apposite istruzioni.

Codice

```
DIM EQU 8
.model small
.stack
.data
opa db ?
opb db ?
ris db ?

.code
.startup
mov ah, 01h        ; codice system call per acquisizione caratteri
xor bx, bx         ; azzeramento bx
mov cx, DIM
```

Codice [cont.]

```
ciclo1: INT 21H ; acquisizione primo operando
    sub al, '0'
    dec cx
    shl al, cl  ; costruzione operando tramite shift e OR
    or  bl, al
    cmp cx, 0
    jnz ciclo1
INT 21h           ; acquisizione CR (suppongo input corretto)
mov opa, bl
xor bx, bx
mov cx, DIM
ciclo2: INT 21H ; acquisizione secondo operando
    sub al, '0'
    dec cx
    shl al, cl
    or bl, al
    cmp cx, 0
    jnz ciclo2
mov opb, bl
```

Codice [cont.]

```
mov al, opb
not al
and al, opa
not al
mov ah, opa
xor ah, opb
or  al, ah

mov ris, al

.exit
end
```

Esercizio 3

- Si scriva un programma in linguaggio Assembly 8086 che conti il numero di bit a 1 nella rappresentazione binaria di una variabile di tipo *byte*.

Codice

```
lung       EQU 8
     .MODEL small
     .STACK
     .DATA
A    DB 7
RES  DB ?
     .CODE
     .STARTUP
     MOV CX, lung
     MOV RES, 0
inizio:  ROL A, 1
     ADC RES, 0
     LOOP inizio
     .EXIT
     END
```

Esercizio 4

- Si scriva un programma in linguaggio Assembly 8086 che esegua una operazione di AND logico tra gli elementi di due vettori di *byte* e ne memorizzi il risultato in un terzo vettore. Il programma deve inoltre contare gli elementi a parità pari nel vettore risultante, cioè tali per cui il numero di 1 nella rappresentazione binaria del numero è pari.

Codice

```
lung       EQU 4
      .MODEL small
      .STACK
      .DATA
A     DB 1, 2, 4, 5
B     DB 3, 4, 1, 3
RES   DB lung dup (?)
PAR   DB 0
      .CODE
      .STARTUP
      LEA BX, RES
      LEA SI, A
      LEA DI, B
      MOV CX, lung
inizio:   MOV AL, [SI]
      AND AL, [DI]
      JNP no_par
      INC PAR
```

Codice [cont.]

```
no_par:   MOV [BX], AL
      INC BX
      INC SI
      INC DI
      LOOP inizio
      .EXIT
      END
```

Esercizio 5

- Si scriva un programma in grado di determinare se ciascuno dei numeri naturali (≥ 2) contenuti in un vettore è primo oppure no. Si ricorda che un numero è primo quando è divisibile solamente per 1 e per se stesso. Siano dati:
  - un vettore di *byte* `numeri` contenente `DIM` elementi (`DIM` dichiarato come costante)
  - un vettore di *byte* `risultato` della stessa dimensione che dovrà contenere, per ogni numero analizzato, un valore logico 1 se il numero nella stessa posizione è primo e 0 se non lo è. Tale vettore sarà modificato dal programma.

Codice

```
DIM     EQU 6
        .MODEL small
        .STACK
        .DATA
numeri  DB 2, 15, 36, 37, 20, 97
ris     DB DIM DUP(?)
        .CODE
        .STARTUP
        LEA SI, numeri
        LEA DI, ris
        MOV CX, DIM
ciclo1: PUSH CX
        MOV BL, [SI]
        CMP BL, 2
        JBE primo ; 2 primo per ipotesi
        MOV CL, BL
        XOR CH, CH
        DEC CX
        MOV BYTE PTR [DI], 0
ciclo2: MOV AL, BL
        XOR AH, AH
        DIV CL
        CMP AH, 0
        JE  next
        DEC CX
        CMP CX, 1
        JNE CICLO2
primo:  MOV BYTE PTR [DI], 1
next:   INC SI
        INC DI
        POP CX
        LOOP ciclo1
        END
```