

Politecnico di Torino - III Facoltà di Ingegneria
Esercitazioni del corso “Sistemi a Microprocessore”
Ing. Paolo Bernardi – email: paolo.bernardi@polito.it

Esercitazione 3 – Le istruzioni per il controllo di flusso

Argomenti trattati:

1. salti incondizionati.
2. salti condizionati dal valore di un flag.
3. salti condizionati dal risultato di un confronto.

Esercizio 1: Si scriva un programma in linguaggio Assembler 8086 che, dato un vettore di variabili di tipo *word* ne inverta il contenuto.

Possibile svolgimento:

```
lung      EQU    6
          .MODEL small
          .STACK
          .DATA
VETT      DW    1, 2, 3, 4, 5, 6
          .CODE
          .STARTUP
          LEA   SI,   VETT
          LEA   DI,   VETT
          MOV   CX,   lung/2
          ADD   DI,   lung*2-2
ciclo:    MOV   AX,   VETT[SI]
          XCHG  VETT[DI], AX
          MOV   VETT[SI], AX
          ADD   SI,   2
          SUB   DI,   2
          LOOP  ciclo
          .EXIT
          END
```

Esercizio 2: Si scriva un programma in linguaggio Assembler 8086 che sommi tutti gli elementi di un vettore definito di tipo *byte* e lo copi in una variabile definita di tipo *word*: se viene generato un riporto sul risultato il programma azzerava la variabile risultato.

Possibile svolgimento:

```
          .MODEL small
          .STACK
          .DATA
VETT      DB    1, 2, 3, 4, 5, 6
RES       DW    ?
          .CODE
          .STARTUP
          LEA   SI,   VETT
          MOV   AX,   0
          MOV   CX,   6
ciclo:    ADD   AL,   [SI]
          ADC   AH,   0
          JO   over
          INC   SI
          LOOP  ciclo
          JMP   fine
over:     MOV   AX,   0
```

```

fine:      MOV    RES, AX
           .EXIT
           END

```

Esercizio 3: Si scriva un programma in linguaggio Assembler 8086 che calcoli il quadrato di un numero su 16 bit utilizzando solo l'istruzione ADD. Il risultato viene memorizzato in una variabile di tipo *word*. Si tenga conto dell'eventuale overflow, azzerando la variabile risultato.

Possibile svolgimento:

```

           .MODEL small
           .STACK
           .DATA
NUM        DW      38000
RES        DW      ?
           .CODE
           .STARTUP
MOV        AX,     NUM
MOV        BX,     NUM
MOV        CX,     NUM
DEC        CX
ciclo:    ADD     AX,  BX
           JO      over
           LOOP   ciclo
           JMP    fine
over:     MOV     AX, 0
fine:     MOV     RES, AX
           .EXIT
           END

```

Esercizio 4: Si scriva un programma in linguaggio Assembler 8086 che trovi il valore max in un vettore di variabili *doubleword* e lo depositi in una variabile di tipo *doubleword*.

Possibile svolgimento:

```

lung      EQU     10
           .MODEL small
           .STACK
           .DATA
VETT      DD     0,1,2,3,4,5,6,7,8,9
MAX       DD     ?
           .CODE
           .STARTUP
MOV        CX,     lung
LEA        SI,     WORD ptr VETT
MOV        AX,     [SI]
MOV        DX,     [SI+2]
lab:      ADD     SI, 4
           CMP     DX, [SI+2]
           JG     no_max
           JNE     max
           CMP     AX, [SI]
           JG     no_max
max:      MOV     AX, [SI]
           MOV     DX, [SI+2]
no_max:   DEC     CX
           JNZ     lab
           MOV     WORD ptr MAX, AX
           MOV     WORD ptr MAX+2, DX
           .EXIT
           END

```

Esercizio 5: Si scriva un programma in linguaggio Assembler 8086 che scriva in un vettore definito di 20 elementi di tipo *word* con i primi 20 valori della serie di Fibonacci.

Serie di Fibonacci: $vet[i] = vet[i-1] + vet[i-2] \Rightarrow vet = 0,1,1,2,3,5,8,13,21,...$

Possibile svolgimento:

```
lung      EQU    20
          .MODEL small
          .STACK
          .DATA
VETT      DW     lung DUP (?)
          .CODE
          .STARTUP
          MOV     CX, lung-2
          MOV     SI, 0
          MOV     VETT[SI], 0
          ADD     SI, 2
          MOV     VETT[SI], 1
lab:      MOV     AX, VETT[SI]
          MOV     BX, VETT[SI-2]
          ADD     AX, BX
          ADD     SI, 2
          MOV     VETT[SI], AX
          LOOP   lab
          .EXIT
          END
```