Corso di Fondamenti di Informatica 2
CdL Ingegneria Informatica
Ing. Franco Zambonelli

ARCHITETTURA DEGLI ELABORATORI: LE ISTRUZIONI dell'8086/8088

Lucidi Realizzati in Collaborazione con:

Prof. Letizia Leonardi
Università di Modena

Prof. Antonio Corradi
Università di Bologna

SET DI ISTRUZIONI

- **istruzioni per il trasferimento dati**
- **istruzioni aritmetiche**
- **istruzioni per la manipolazione di bit**
- **istruzioni per il trasferimento del controllo**
- **istruzioni che operano sulle stringhe**
- **istruzioni per il controllo del processore**

Istruzioni per il trasferimento dati

GENERAL PURPOSE	
MOV destinazione, sorgente	Muove un byte o una word
PUSH sorgente	Push di una word nello stack
POP destinazione	Pop di una word dallo stack
XCHG destinazione, sorgente	Scambia un byte o una word
XLATtabella-sorgente	Traduce un byte
INPUT/OUTPUT	
IN accumulatore, porta	Input di un byte o una word
OUT porta, accumulatore	Output di un byte o una word
TRASFERIMENTO di INDIRIZZI	
LEA reg16, mem16	Carica un indirizzo effettivo (EA)
LDS reg16, mem32	Carica un indirizzo completo in DS:reg16
LES reg16, mem32	Carica un indirizzo completo in ES:reg16
TRASFERIMENTO di FLAG	
LAHF	Carica AH con il registro flag (0-7)
SAHF	Carica il registro flag (0-7) con AH
PUSHF	Push del registro flag nello stack
POPF	Pop del registro flag dallo stack

MOV destinazione, sorgente

Istruzione di trasferimento generico

Trasferisce un byte o una word
- **tra due registri**
- **tra un registro e una locazione di memoria**

Sorgente può essere una costante

Per trasferire un dato da una locazione di memoria a un'altra:

```
MOV  AX, fromVar     ;  copia fromVar in AX
MOV  toVar, AX   ;  e quindi in toVar
```

Non è possibile caricare una costante (come l'indirizzo di un segmento) **in un registro di segmento**:

```
MOV  AX, segData     ;  indirizzo di segData in AX
MOV  DS, AX          ;  e quindi in DS
```

Il registro CS non è utilizzabile come destinazione

PUSH sorgente
POP destinazione

push e **pop al top dello stack selezionato da SS:SP**

L'unico operando (sempre una word)
può essere **un registro o una locazione di memoria**
NON può essere **una costante**

```
MOV    AX, 100            ;  costante in AX
PUSH   AX                 ;  e quindi sullo stack

MOV    AX, OFFSET Var     ;  prima in AX
PUSH   AX                 ;  e quindi sullo stack
```

PUSHF (PUSH Flags onto stack)
POPF (POP Flags off stack)

Trasferiscono il contenuto del registro flag (16 bit) nello stack e viceversa

PUSHF ≡ PUSH del registro flag

POPF ≡ POP del registro flag

Necessarie in quanto il registro flag non ha un nome

XCHG destinazione, sorgente

Scambia (eXCHanGe) il contenuto (byte o word) dei due operandi - due registri o un registro e una locazione di memoria

Per scambiare il contenuto di due locazioni di memoria:

```
XCHG   AX, var1
XCHG   AX, var2       ;  var2 <- var1
XCHG   AX, var1       ;  var1 <- var2
```

oppure:

```
PUSH  var1
PUSH  var2
POP   var1
POP   var2
```

LEA reg16, mem16

LEA (Load Effective Address):

- **calcola l'EA del secondo operando** (un riferimento a una word in memoria) **e**
- **lo carica nel primo operando** (un registro generale a 16 bit)

```
LEA  BX, Table     ; MOV  BX, OFFSET Table

LEA  BX, [Table + BP + SI]
```

il secondo esempio non ha un'istruzione MOV equivalente, in quanto l'operatore OFFSET richiede un'espressione costante

LDS reg16, mem32
LES reg16, mem32

LDS (Load pointer using DS) e LES (Load pointer using ES)

- **leggono un indirizzo completo dalla memoria** (segmento + offset)
- **memorizzano l'indirizzo del segmento nel corrispondente registro segmento** (DS o ES)
- **memorizzano i 16 bit dell'offset nel registro selezionato**

```
Table      DW    100 DUP (0)
AddrTable  DD    Table    ; segmento:offset di Table
           . . .
           LDS   BX, AddrTable
```

DS <- indirizzo del segmento contenente Table
BX <- offset di Table nel segmento

Sequenza equivalente (LDS):

```
MOV   BX, AddrTable
MOV   DS, AddrTable + 2
```

Sequenza semi-equivalente (LDS):

```
MOV   BX, OFFSET Table
MOV   AX, SEG Table
MOV   DS, AX
```

Istruzioni aritmetiche

ADDIZIONE	
ADD destinazione, sorgente	Somma byte o word
ADC destinazione, sorgente	Somma byte o word con riporto (carry)
INC destinazione	Incrementa byte o word di 1
AAA	ASCII adjust per addizione
DAA	Decimal adjust per addizione
SOTTRAZIONE	
SUB destinazione, sorgente	Sottrae byte o word
SBB destinazione, sorgente	Sottrae byte o word con riporto (borrow)
DEC destinazione	Decrementa byte o word di 1
NEG destinazione	Cambia segno a byte o word
CMP destinazione, sorgente	Confronta byte o word
AAS	ASCII adjust per sottrazione
DAS	Decimal adjust per sottrazione
MOLTIPLICAZIONE	
MUL sorgente	Moltiplica byte o word senza segno
IMUL sorgente	Moltiplica byte o word con segno
AAM	ASCII adjust per moltiplicazione
DIVISIONE	
DIV sorgente	Divide byte o word senza segno
IDIV sorgente	Divide byte o word con segno
AAD	ASCII adjust per divisione
ESTENSIONE DEL SEGNO	
CBW	Converte byte in word
CWD	Converte word in doubleword

ADD destinazione, sorgente
ADC destinazione, sorgente

Sommano 2 operandi di 8 o 16 bit

ADD somma l'operando sorgente all'operando destinazione e memorizza il risultato nell'operando destinazione:

destinazione += sorgente

ADC (ADd with Carry) effettua la stessa operazione, comprendendo nella somma il flag del riporto (CF):

destinazione += sorgente + CF

L'istruzione ADC permette di effettuare la somma di numeri maggiori di 16 bit

Somma di due numeri di 32 bit
(AX:BX) <- (AX:BX) + (CX:DX)

```
ADD  BX, DX    ; somma i 16 bit meno significativi
ADC  AX, CX    ; somma i 16 bit più significativi
```

Somma di due numeri di 64 bit

VeryLong1 += VeryLong2

```
MOV  AX, VeryLong2
ADD  VeryLong1, AX          ; somma i bit da 0 a 15
MOV  AX, [VeryLong2 + 2]
ADC  [VeryLong1 + 2], AX    ; somma i bit da 16 a 31
MOV  AX, [VeryLong2 + 4]
ADC  [VeryLong1 + 4], AX    ; somma i bit da 32 a 47
MOV  AX, [VeryLong2 + 6]
ADC  [VeryLong1 + 6], AX    ; somma i bit da 48 a 63
```

L'8086 memorizza gli interi ponendo il byte meno significativo all'indirizzo più basso di memoria

Le istruzioni di somma modificano il contenuto dei seguenti flag:

- **CF = (esiste un riporto) ? 1 : 0**
- **OF = (overflow) ? 1 : 0**
- **SF = segno del risultato**
- **ZF = (il risultato è zero) ? 1 : 0**

inoltre vengono modificati i flag PF e AF

SUB destinazione, sorgente
SBB destinazione, sorgente

SUB (SUBtract) sottrae l'operando sorgente all'operando destinazione e memorizza il risultato nell'operando destinazione:

destinazione –= sorgente

SBB (SuBtract with Borrow) effettua la stessa operazione, comprendendo nella sottrazione il flag del riporto (CF):

destinazione –= sorgente + CF

L'istruzione SBB permette di effettuare la sottrazione di numeri maggiori di 16 bit

Sottrazione di due numeri di 32 bit
(AX:BX) <- (AX:BX) – (CX:DX)

```
SUB  BX, DX    ; sottrae i 16 bit meno significativi
SBB  AX, CX    ; sottrae i 16 bit più significativi
```

Le istruzioni di sottrazione modificano il contenuto dei **flag** in modo analogo alle istruzioni di somma

INC destinazione

Incrementa di 1 il valore del suo operando

DEC destinazione

Decrementa di 1 il valore del suo operando

NEG destinazione

NEG (NEGate) cambia segno al suo operando
complemento a 2

CMP destinazione, sorgente

CMP (CoMPare) agisce esattamente come l'istruzione SUB, senza però modificare l'operando destinazione

Lo scopo è quello di modificare il valore dei flag di stato (ved. SUB)

MUL sorgente (MULtiply, unsigned)
IMUL sorgente (Integer MULtiply, signed)

Se l'operando è **un byte**, viene eseguito:

```
            AX = AL * sorgente;
se MUL:     CF = OF = (AH == 0) ? 0 : 1;
se IMUL:    CF = OF = (AH == 0xFF && AL[7] == 1
              || AH == 0x00 && AL[7] == 0) ? 0 : 1;
```

Se l'operando è **una word**, viene eseguito:

```
            DX:AX = AX * sorgente;
se MUL:     CF = OF = (DX == 0) ? 0 : 1;
se IMUL:    CF = OF = (DX == 0xFFFF && AX[15] == 1
              || DX == 0x0000 && AX[15] == 0) ? 0 : 1;
```

L'operando sorgente non può essere una costante
per moltiplicare AX per 100 è necessario scrivere:

```
MOV  DX, 100
MUL  DX
```

DIV sorgente (DIVide, unsigned)
IDIV sorgente (Integer DIVide, signed)

se l'operando è **un byte**, viene eseguito:

AL = Quoziente(AX / sorgente);
AH = Resto(AX / sorgente);

se l'operando è **una word**, viene eseguito:

AX = Quoziente(DX:AX / sorgente);
DX = Resto(DX:AX / sorgente);

L'operando sorgente non può essere una costante
per dividere DX:AX per 100 è necessario scrivere:

```
MOV  BX, 100
DIV  BX
```

Le istruzioni di divisione lasciano indefiniti i flag di stato
Se il **quoziente eccede la capacità** del registro destinazione o se il **divisore è zero**, viene generato un **interrupt di tipo 0** (divide by 0)

CBW
CWD

CBW (Convert Byte to Word) converte il byte con segno contenuto in AL in una word con segno contenuta in AX, estendendo il bit del segno in tutti i bit di AH

AH = (AL[7] == 1) ? 0xFF : 0x00;

CWD (Convert Word to Doubleword) converte una word con segno contenuta in AX in una doubleword con segno contenuta in DX:AX, estendendo il bit del segno in tutti i bit di DX

DX = (AX[15] == 1) ? 0xFFFF : 0x0000;

Istruzioni per la manipolazione di bit

OPERAZIONI LOGICHE	
AND destinazione, sorgente	AND di byte o word
OR destinazione, sorgente	OR di byte o word
XOR destinazione, sorgente	XOR di byte o word
NOT destinazione	NOT di byte o word
TEST destinazione, sorgente	TEST di byte o word
SHIFT	
SAL destinazione, cont	Shift aritmetico di byte o word a sinistra
SAR destinazione, cont	Shift aritmetico di byte o word a destra
SHL destinazione, cont	Shift logico di byte o word a sinistra
SHR destinazione, cont	Shift logico di byte o word a destra
ROTAZIONE	
ROL destinazione, cont	Ruota byte o word a sinistra
ROR destinazione, cont	Ruota byte o word a destra
RCL destinazione, cont	Ruota byte o word a sinistra con carry
RCR destinazione, cont	Ruota byte o word a destra con carry

AND destinazione, sorgente
OR destinazione, sorgente
XOR destinazione, sorgente

Eseguono le corrispondenti operazioni logiche bit a bit:

```
AND     dst, src  ;  dst &= src
OR      dst, src  ;  dst |= src
XOR     dst, src  ;  dst ^= src
```

Le istruzioni pongono a zero i flag OF e CF e modificano opportunamente i flag SF, ZF e PF

NOT destinazione

Complementa tutti i bit del suo operando
complemento a 1

TEST destinazione, sorgente

TEST agisce come l'istruzione AND, senza però modificare l'operando destinazione

Lo scopo è quello di modificare il valore dei flag di stato (ved. AND)

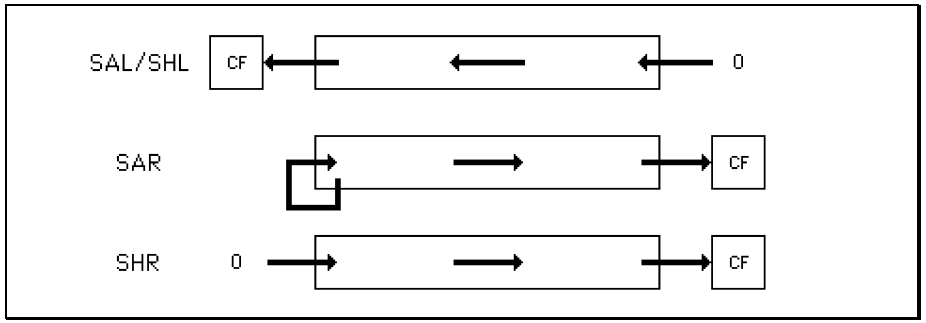
SAL destinazione, contatore
SAR destinazione, contatore
SHL destinazione, contatore
SHR destinazione, contatore

SAL (Shift Arithmetic Left) shift verso sinistra di intero con segno

SAR (Shift Arithmetic Right) shift verso destra di intero con segno

SHL (SHift logical Left) shift verso sinistra di intero senza segno

SHR (SHift logical Right) shift verso destra di intero senza segno



Contatore può assumere solo i valori 1 o CL
in quest'ultimo caso, il registro CL deve contenere il numero di shift da eseguire

SAL e SHL possono essere utilizzate per effettuare **moltiplicazioni per potenze di 2** di numeri con o senza segno
SAR e SHR possono essere utilizzate per effettuare **divisioni per potenze di 2** di numeri con o senza segno

8 clock (4 x 2):

```
SAL  AX, 1        ;  AX * 2
SAL  AX, 1        ;  AX * 4
SAL  AX, 1        ;  AX * 8
SAL  AX, 1        ;  AX * 16
```

28 clock (4 per MOV e 8 + 4 x 4 per SAL):

```
MOV  CL, 4        ;  la potenza di 2
SAL   AX, CL      ;  AX * 16
```

122 clock (4 per MOV e 118 per MUL)

```
MOV  CX, 16
MUL  CX           ;  DX:AX <- AX * 16
```

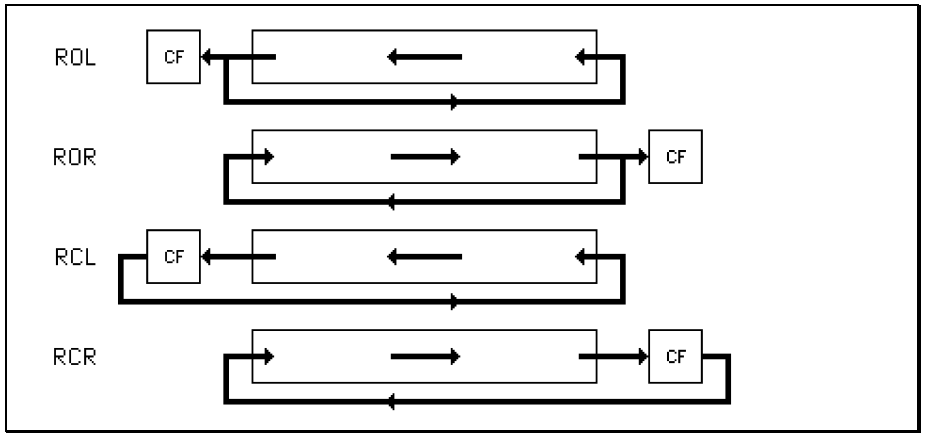
ROL destinazione, contatore
ROR destinazione, contatore
RCL destinazione, contatore
RCR destinazione, contatore

ROL (ROtate Left) ruota il primo operando verso sinistra

ROR (ROtate Right) ruota il primo operando verso destra

RCL (Rotate Left through Carry) ruota il primo operando e CF verso sinistra

RCR (Rotate Right through Carry) ruota il primo operando e CF verso destra



contatore può assumere solo i valori 1 o CL

Istruzioni per il trasferimento del controllo

Modificano (eventualmente sotto condizione) il contenuto dei registri CS e IP

CS viene modificato solo se il controllo viene trasferito in un segmento di codice diverso da quello di partenza

TRASFERIMENTO INCONDIZIONATO	
CALLtarget	Chiama una procedura
RET [pop-value]	Ritorna da una procedura
JMP target	Salta
TRASFERIMENTO CONDIZIONATO	
Jxxx short-label	Salta sotto condizione
JCXZshort-label	Salta se il registro CX = 0
CONTROLLO delle ITERAZIONI	
LOOP short-label	Cicla
LOOPE short-label	Cicla se uguale
LOOPNE short-label	Cicla se non uguale
LOOPZ short-label	Cicla se zero
LOOPNZ short-label	Cicla se diverso da zero
INTERRUPT	
INT tipo-di-interrupt	Interrupt
INTO	Interrupt se overflow
IRET	Ritorna da una procedura di interr

CALL procedura
RET

CALL trasferisce il controllo dal programma chiamante alla procedura chiamata
- salva l'indirizzo di ritorno sullo stack
- passa il controllo alla procedura chiamata

RET trasferisce il controllo dalla procedura chiamata al programma chiamante
- legge dallo stack l'indirizzo di ritorno salvato dalla corrispondente CALL
- ripassa il controllo al chiamante

Trasferimenti del controllo:
- nell'ambito dello stesso segmento
- in segmenti distinti

Se la procedura chiamata è **NEAR**:
- CALL esegue **push IP**
- RET esegue **pop IP**

Se la procedura chiamata è **FAR**:
- CALL esegue **push CS** e **push IP**
- RET esegue **pop IP** e **pop CS**

Si noti che, al momento della definizione di una procedura, **gli attributi NEAR e FAR sono necessari** in quanto l'assemblatore non è in grado di determinare quale tipo di istruzione RET utilizzare all'interno della procedura

Chiamate nidificate a procedure
- ogni CALL inserisce nello stack un indirizzo di ritorno **di 2 o 4 byte**
- il livello di nidificazione è limitato dallo spazio disponibile sullo stack

Chiamata a procedura **diretta**:

```
    CALL    FarProc
    CALL    NearProc
```

Chiamata a procedura **indiretta**, cioè tramite **un registro o una locazione di memoria** che contiene l'indirizzo della procedura da chiamare:

```
    LEA     BX,  NearProc
    CALL    BX

NAdd        DW NearProc
    . . .
    CALL    NAdd

FAdd        DD  FarProc
    . . .
    CALL    Fadd


    CALL    [BX];  ?
    CALL    WORD  PTR  [BX]     ;  Near call
            ;  BX -> offset
    CALL    DWORD  PTR  [BX]    ;  Far call
            ;  BX -> seg:offset
```


JMP label

Trasferisce il controllo all'istruzione specificata dall'operando, in modo incondizionato

```
    JMP  Label1
    . . .
Label1:
    . . .
```

Nell'istruzione macchina displacement relativo di 16 bit o di 8 bit (short)

```
IP <- IP + disp                                 (se di 16 bit)
IP <- IP + disp sign extended a 16 bit          (se di 8 bit)
```

SHORT espressione

Operatore che informa l'assemblatore che espressione è distante dalla locazione corrente da -128 a +127 byte

espressione è **un riferimento a una label nel segmento codice corrente**

```
    JMP  short Label1
    . . .
Label1:
    . . .
```

Jxxx short-label
JCXZ short-label (Jump if CX is Zero)

Le istruzioni per il trasferimento condizionato controllano se una condizione è verificata:

- **se la condizione è verificata** ⇒
  il controllo passa all'istruzione di label "short-label"
- **se la condizione non è verificata** ⇒
  l'esecuzione prosegue con la successiva istruzione

Nell'istruzione macchina displacement relativo
sempre di 8 bit

short-label deve fare riferimento a una istruzione:

- contenuta nello **stesso segmento di codice** dell'istruzione di salto
- a una **distanza in byte** (displacement) rispetto all'istruzione di salto **compresa tra -128 e +127**

Istruzione	Descrizione	Salta se ...
JA	Jump if Above	CF = 0 e ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 o ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1
JG	Jump if Greater	ZF = 0 e SF = OF
JGE	Jump if Greater or Equal	SF = OF
JL	Jump if Less	SF • OF
JLE	Jump if Less or Equal	ZF = 1 o SF • OF
JNA	Jump if Not Above	CF = 1 o ZF = 1
JNAE	Jump if Not Above nor Equal	CF = 1
JNB	Jump if Not Below	CF = 0
JNBE	Jump if Not Below nor Equal	CF = 0 e ZF = 0
JNC	Jump if No Carry	CF = 0
JNE	Jump if Not Equal	ZF = 0
JNG	Jump if Not Greater	ZF = 1 o SF • OF
JNGE	Jump if Not Greater nor Equal	SF • OF
JNL	Jump if Not Less	SF = OF
JNLE	Jump if Not Less nor Equal	ZF = 0 e SF = OF
JNO	Jump if No Overflow	OF= 0
JNP	Jump if No Parity (odd)	PF = 0
JNS	Jump if No Sign	SF = 0
JNZ	Jump if Not Zero	ZF = 0
JO	Jump on Overflow	OF =1
JP	Jump on Parity (even)	PF = 1
JPE	Jump if Parity Even	PF = 1
JPO	Jump if Parity Odd	PF = 0
JS	Jump on Sign	SF = 1
JZ	Jump if Zero	ZF = 1

Esempi di salti condizionati:

```
ADD  AX, BX
JC  LabelOverflow
```

JC trasferisce il controllo all'istruzione di label LabelOverflow se la somma dei registri AX e BX ha prodotto un riporto (CF = 1)

```
SUB  AX, BX
JZ  LabelZero
```

JZ trasferisce il controllo all'istruzione di label LabelZero se la differenza dei registri AX e BX è nulla (ZF = 1)
Per controllare se AX e BX contengono lo stesso valore, senza dover effettuare la differenza, si utilizza l'istruzione CMP:

```
CMP  AX, BX
JE  LabelZero
```

si noti l'utilizzo di **JE al posto di JZ**:
le due istruzioni sono del tutto equivalenti, ma, dato che nell'ultima sequenza si testa un uguaglianza, è più chiaro usare JE

Per alcuni tipi di test è necessario scegliere tra **due differenti istruzioni di salto**, a seconda che si stia testando il risultato di un'operazione tra **valori con o senza segno**

Istruzioni di salto da utilizzare subito dopo l'istruzione CMP, in funzione:
- del tipo di test che si desidera effettuare
- del tipo di dato confrontato

Per saltare se	Valori senza segno	Valori con segno
Destinazione = Sorgente	JE	JE
Destinazione • Sorgente	JNE	JNE
Destinazione > Sorgente	JA	JG
Destinazione • Sorgente	JAE	JGE
Destinazione < Sorgente	JB	JL
Destinazione • Sorgente	JBE	JLE

```
    CMP  AX, BX
    JAE  LabelGreaterOrEqual
    . . .       ; istruzioni eseguite se AX < BX
LabelGreaterOrEqual:
    JA LabelGreater
    . . .       ; istruzioni eseguite se AX = BX
LabelGreater:
    . . .       ; istruzioni eseguite se AX > BX
```

esegue **tre differenti blocchi di istruzioni**, in funzione dei valori **senza segno** contenuti nei registri AX e BX
nel caso di valori **con segno**,
sostituire JAE con JGE e JA con JG

Trasferimento del controllo a una label
la cui distanza è superiore a quella ammessa

```
    JZ  FarLabel
```

deve essere sostituita con la sequenza:

```
    JNZ  NearLabel   ; test invertito
    JMP  FarLabel    ; salto incondizionato se
                     ; JZ verificato
NearLabel:
    . . .
```


LOOP short-label

Istruzione per il controllo delle iterazioni
- decrementa di 1 il registro CX
- in base al risultato, decide se:
  - saltare all'istruzione specificata dall'operando
  - continuare con l'istruzione successiva

LOOP decrementa di 1 il registro CX e trasferisce il controllo al suo operando se CX è diverso da zero

```
        MOV  CX, 100     ;  CX <- Numero di iterazioni
Start:
        . . .
        LOOP  Start      ;  se CX diverso da zero
                         ;  salta a Start
        . . .            ;  altrimenti continua
```

esegue 100 volte il blocco di istruzioni tra la label Start e l'istruzione LOOP

LOOPE (LOOP if Equal)
LOOPNE (LOOP if Not Equal),
LOOPZ (LOOP if Zero)
LOOPNZ (LOOP if Not Zero)

LOOPE e LOOPZ si comportano in modo identico:
- **decrementano CX**
- **saltano alla short-label se è soddisfatta la condizione CX ≠ 0 and ZF = 1**

si esce dal ciclo
- quando il contatore è a zero, oppure
- quando viene azzerato ZF

LOOPNE e LOOPNZ si comportano in modo identico:
- **decrementano CX**
- **saltano alla short-label se è soddisfatta la condizione CX ≠ 0 and ZF = 0**

si esce dal ciclo
- quando il contatore è a zero, oppure
- quando viene posto a 1 il flag ZF

INT tipo-di-interrupt
IRET

Un interrupt è simile a una chiamata a procedura

Mentre una chiamata a procedura può essere NEAR o FAR e diretta o indiretta,
l'interrupt esegue sempre una chiamata FAR indiretta, **prendendo l'indirizzo della routine (di servizio) nel vettore degli interrupt**

Mentre una chiamata a procedura salva nello stack l'indirizzo di ritorno,
l'interrupt salva nello stack, oltre all'indirizzo di ritorno, anche il registro flag

INT effettua le seguenti azioni:

1. **push del registro flag nello stack** e **azzeramento dei flag TF e IF** al fine di disabilitare il single-step e il riconoscimento di altri interrupt mascherabili

2. **lettura dell'indirizzo** (di 32 bit) **della routine di servizio dell'interrupt** dalla tabella degli interrupt (la posizione in tabella è data dal tipo di interrupt)

3. **push dei registri CS, IP nello stack** e **caricamento in CS, IP del nuovo indirizzo** (come per CALL di tipo FAR)

IRET deve essere utilizzata in ogni routine di servizio degli interrupt al fine di restituire correttamente il controllo al programma che era stato interrotto (via software, o via hardware)

IRET esegue la **pop di tre word dallo stack** e **le carica**, rispettivamente, **in IP, in CS e nel registro flag**

Le routine di servizio degli interrupt sono strutturalmente delle normali procedure, però
non possono contenere istruzioni RET,
così come le normali procedure non possono contenere istruzioni IRET

Netta distinzione tra i due tipi di procedure:
- una routine di servizio degli interrupt non potrà mai essere chiamata con una CALL
- una normale procedura non potrà mai essere chiamata con un INT
- ad ogni CALL deve corrispondere un RET
- ad ogni INT deve corrispondere un IRET

Routine di servizio e procedure normali possono essere nidificate in tutti i modi possibili

INTO (INTerrupt if Overflow)

Se OF == 1, INTO corrisponde a un INT 4, in caso contrario, l'esecuzione procede normalmente

```
      JNO  skipLabel   ;  salta se OF = 0
      INT  4           ;  in caso contrario INT 4
skipLabel:             ;  convergenza
```

Istruzioni
per il controllo del processore

OPERAZIONI sui FLAG	
STC	Setta il flag di riporto (CF)
CLC	Azzera il flag di riporto (CF)
CMC	Complementa il flag di riporto (CF)
STD	Setta il flag di direzione (DF)
CLD	Azzera il flag di direzione (DF)
STI	Setta il flag di interrupt enable (IF)
CLI	Azzera il flag di interrupt enable (IF)
SINCRONIZZAZIONE ESTERNA	
HLT	Aspetta un reset o un interrupt
WAIT	Aspetta un segnale esterno su linea TEST
ESC ext-opcode, sorgente	Invia un'istruzione a un processore esterno (ad esempio, 8087)
LOCK (prefisso)	Blocca il bus durante l'esecuzione dell'istruzione associata
NO OPERATION	
NOP	Nessuna operazione