Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

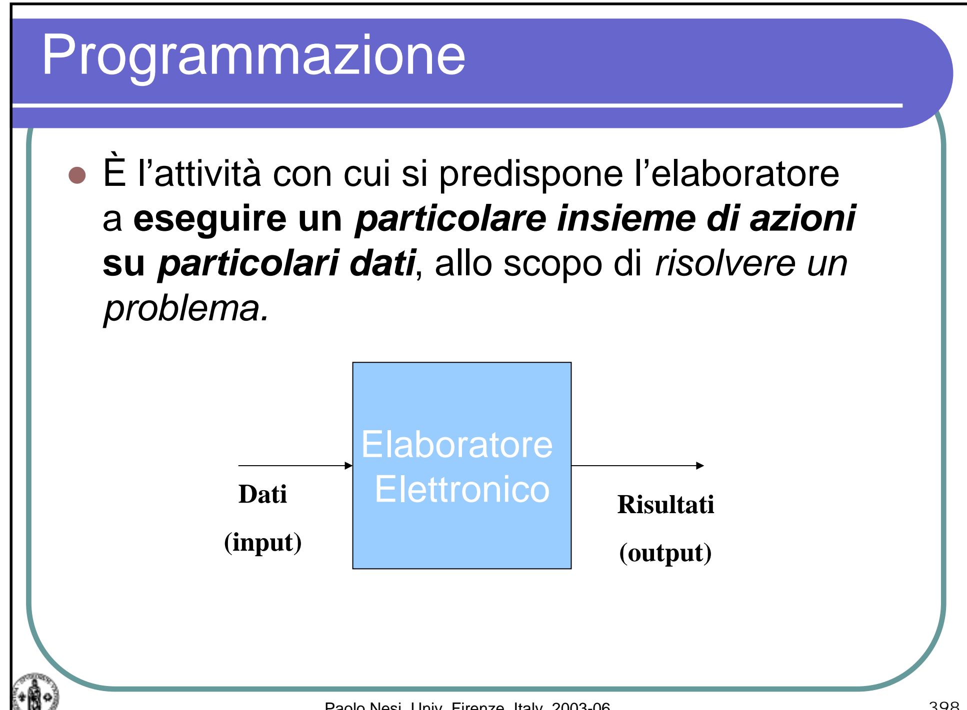
Nuovo Ordinamento

Parte 8bis, La Programmazione

Dr. Ing. Ivan Bruno
Prof. Paolo Nesi
http://www.dsi.unifi.it/~ivanb
ivanb@dsi.unifi.it
2006

Introduzione alla programmazione

- Il calcolatore elettronico è uno strumento in grado di eseguire insiemi di *azioni* (“*mosse*”) *elementari*
- le azioni vengono eseguite su oggetti (***dati***) per produrre altri oggetti (***risultati***)
- l’esecuzione di azioni viene richiesta all’elaboratore attraverso *frasi* scritte in qualche *linguaggio* (*istruzioni*)



Problemi da risolvere

- I problemi che siamo interessati a risolvere con l'elaboratore sono di natura molto varia:
  - *Dati due numeri trovare il maggiore*
  - *Dato un elenco di nomi e relativi numeri di telefono trovare il numero di telefono di una determinata persona*
  - *Dati a e b, risolvere l'equazione ax+b=0*
  - *Stabilire se una parola viene alfabeticamente prima di un'altra*
  - *Somma di due numeri interi*
  - *Scrivere tutti gli n per cui l'equazione:* $X^n+Y^n = Z^n$ *ha soluzioni intere (problema di Fermat)*
  - *Ordinare una lista di elementi*
  - *Calcolare il massimo comun divisore fra due numeri dati.*
  - *Calcolare il massimo in un insieme.*
  - *….*

Paolo Nesi, Univ. Firenze, Italy, 2003-06 399

Risoluzione dei problemi

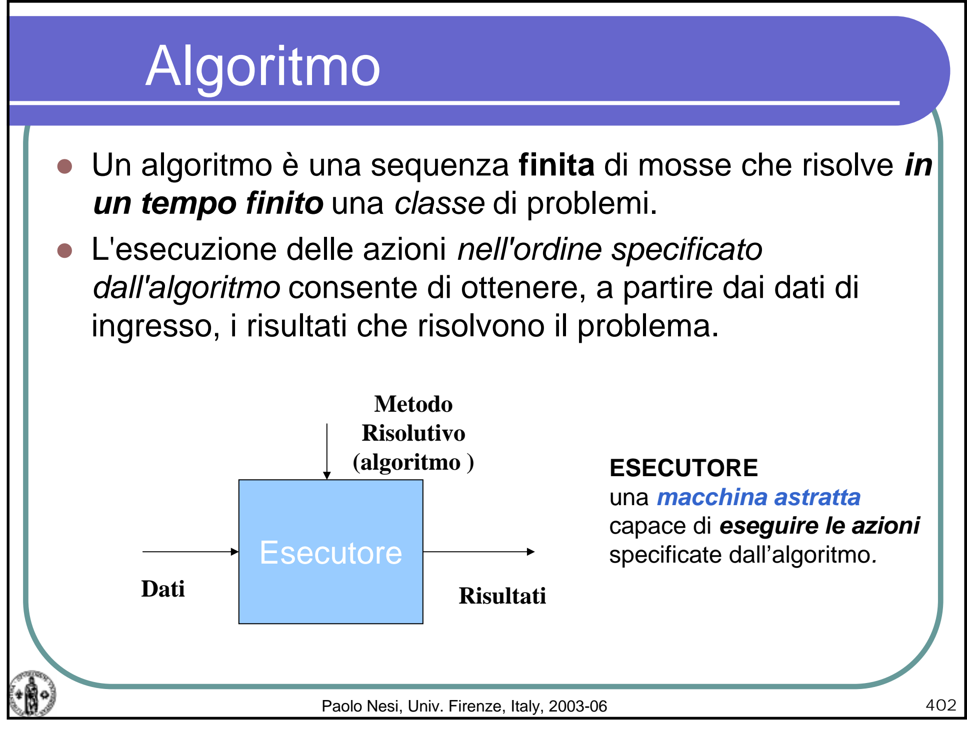
- La descrizione del problema non fornisce (in generale) un metodo per risolverlo.
  - Affinché un problema sia risolvibile è però necessario che la sua definizione sia chiara e completa
- Non tutti i problemi sono risolvibili attraverso l'uso del calcolatore. Esistono classi di problemi per le quali la soluzione automatica non è proponibile. Ad esempio:
  - se il problema presenta infinite soluzioni
  - per alcuni dei problemi **non è stato trovato** un metodo risolutivo
  - per alcuni problemi è stato dimostrato che **non esiste** un metodo risolutivo automatizzabile

Paolo Nesi, Univ. Firenze, Italy, 2003-06 400

Risoluzione dei problemi

- *La risoluzione di un problema è il processo che, dato un problema e individuato un opportuno metodo risolutivo, trasforma i dati iniziali nei corrispondenti risultati finali.*
- Affinché la risoluzione di un problema possa essere realizzata attraverso l'uso del calcolatore, tale processo deve poter essere definito come *sequenza di azioni elementari.*

Paolo Nesi, Univ. Firenze, Italy, 2003-06 401



Algoritmi: proprietà

- **Eseguibilità**: ogni azione dev'essere *eseguibile* dall'esecutore *in un tempo finito*.
- **Non ambiguità**: ogni azione deve essere *univocamente interpretabile* dall'esecutore.
- **Finitezza**: il numero totale di azioni da eseguire, per ogni insieme di dati di ingresso, deve essere finito.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 403

Algoritmi: proprietà (2)

- **Quindi, l'algoritmo deve:**
  - essere *applicabile a qualsiasi insieme di dati di ingresso* appartenenti al **dominio di definizione** dell'algoritmo
  - essere costituito da operazioni appartenenti ad un determinato **insieme di operazioni fondamentali**
  - essere costituito da **regole non ambigue**, cioè interpretabili in modo **univoco** qualunque sia l'esecutore (persona o "macchina") che le legge.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 404

Algoritmi e programmi

- Ogni elaboratore è una macchina in grado di eseguire azioni elementari su oggetti detti **DATI**.
- L'esecuzione delle azioni è richiesta all'elaboratore tramite comandi elementari chiamati **ISTRUZIONI** espresse attraverso un opportuno formalismo: il **LINGUAGGIO di PROGRAMMAZIONE**.
- La formulazione testuale di un algoritmo in un linguaggio comprensibile a un elaboratore è detta **programma**.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 405

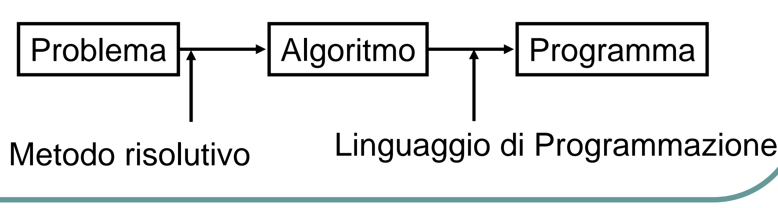
Programma

- **Un programma è un testo** scritto in accordo alla **sintassi** e alla **semantica** di un linguaggio di programmazione.
- Un ***programma*** è la **formulazione testuale,** in un certo linguaggio di programmazione, di un **algoritmo** che risolve un dato *problema*.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 406

Algoritmo & programma

- Passi per la risoluzione di un problema:
  - individuazione di un procedimento risolutivo
  - scomposizione del procedimento in un insieme ordinato di azioni ⇒ **ALGORITMO**
  - rappresentazione dei dati e dell'algoritmo attraverso un formalismo comprensibile dal calcolatore ⇒ **LINGUAGGIO DI PROGRAMMAZIONE**



Paolo Nesi, Univ. Firenze, Italy, 2003-06 407

Algoritmi equivalenti

- Due algoritmi si dicono ***equivalenti*** quando:
  - hanno lo stesso **dominio di ingresso**;
  - hanno lo stesso **dominio di uscita**;
  - in corrispondenza degli **stessi valori del dominio di ingresso *producono gli stessi valori* nel dominio di uscita.**
- Due algoritmi ***equivalenti***
  - forniscono lo **stesso risultato**
  - ma possono avere ***diversa efficienza***
  - e possono essere **profondamente diversi!**

Paolo Nesi, Univ. Firenze, Italy, 2003-06 408

Algoritmi: esempi

- **Soluzione dell'equazione ax+b=0**
  - leggi i valori di a e b
  - calcola -b
  - dividi quello che hai ottenuto per a e chiama x il risultato
  - stampa x
- **Calcolo del massimo di un insieme**
  - Scegli un elemento come massimo provvisorio *max*
  - Per ogni elemento *i* dell'insieme: se *i*>*max* eleggi *i* come nuovo massimo provvisorio *max*
  - Il risultato è *max*

Paolo Nesi, Univ. Firenze, Italy, 2003-06 409

Algoritmi: esempi

- **Stabilire se una parola P viene alfabeticamente prima di una parola Q**
- **leggi** P,Q
- **ripeti quanto segue:**
  - **se** prima lettera di P < prima lettera di Q
  - **allora** scrivi vero
  - **altrimenti se** prima lettera P > Q
    - **allora** scrivi falso
    - **altrimenti** (le lettere sono =) togli da P e Q la prima lettera
- **fino** a quando hai trovato le prime lettere diverse.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 410

Linguaggio & Programma

- Dato un algoritmo, un **programma è la sua descrizione *in un particolare linguaggio*** di programmazione
- **Un linguaggio di programmazione è una notazione formale** che può essere usata per descrivere algoritmi.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 411

Linguaggi di Programmazione

I linguaggi sono classificati in:

- Low level
  - ⇒ Linguaggio macchina & Linguaggio Assembly
- High level
  - ⇒ C/C++, Pascal, Delphi, Java, Cobol….

Linguaggi di Programmazione

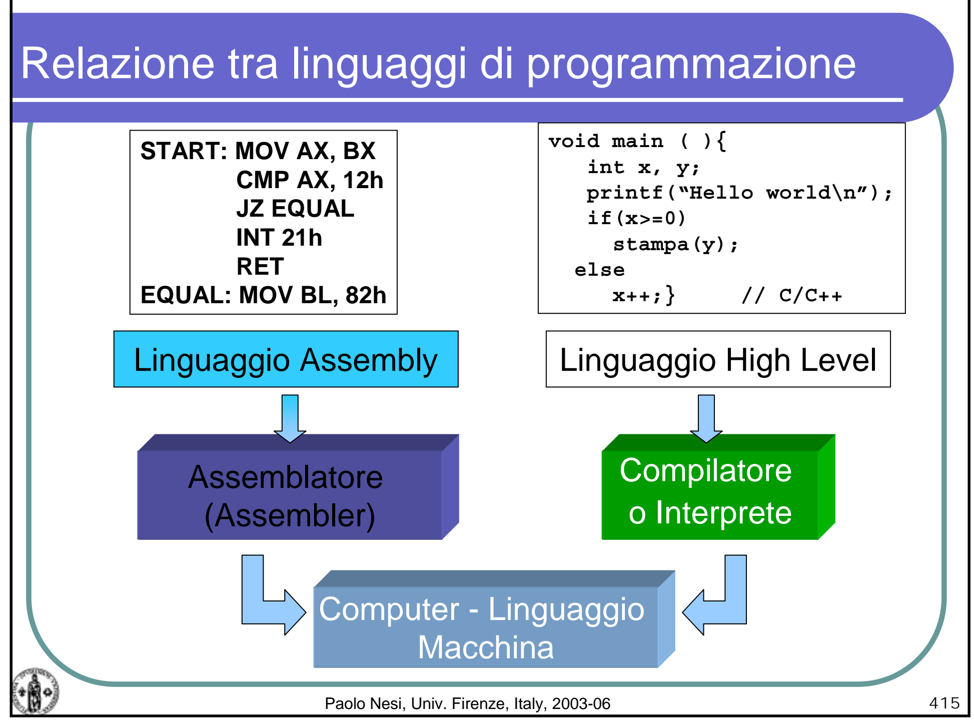
- I linguaggi low level sono strettamente legati all'architettura interna del computer
- Ad ogni singola azione corrisponde una linea di programma (rapporto 1 a 1)
- I linguaggi high level sono più "vicini" al nostro modo di pensare
- Devono essere opportunamente tradotti per essere compresi da un computer

Assembly

- Il linguaggio naturale di un computer è il *linguaggio macchina*
- Le istruzioni del linguaggio macchina sono rappresentate da una sequenza binaria di '1' e '0'
- Il linguaggio *Assembly* è il linguaggio più vicino alla macchina ma deve essere tradotto in *linguaggio macchina* per essere eseguito (*Assembler*)
- Processori diversi hanno differenti linguaggi macchina e perciò necessitano di un proprio linguaggio assembly.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 414

Relazione tra linguaggi di programmazione



Sintassi e semantica

Aspetti caratteristici del linguaggio:

- Esistenza di un insieme di **parole chiave**
- Sintassi: **l'insieme di regole formali per la scrittura di programmi** in un linguaggio, che dettano le ***modalità per costruire frasi corrette*** nel linguaggio stesso.
- Semantica: **l'insieme dei significati** da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio.

Compilatore vs Interprete

- I **compilatori** traducono automaticamente un programma dal linguaggio *L* a quello macchina (per un determinato elaboratore).
  - Durante la traduzione verificano la correttezza di ciascuna istruzione.
  - La traduzione termina quando non ci sono più errori sintattici.
  - Al termine della traduzione il programma è pronto per essere eseguito.

Compilatore vs Interprete

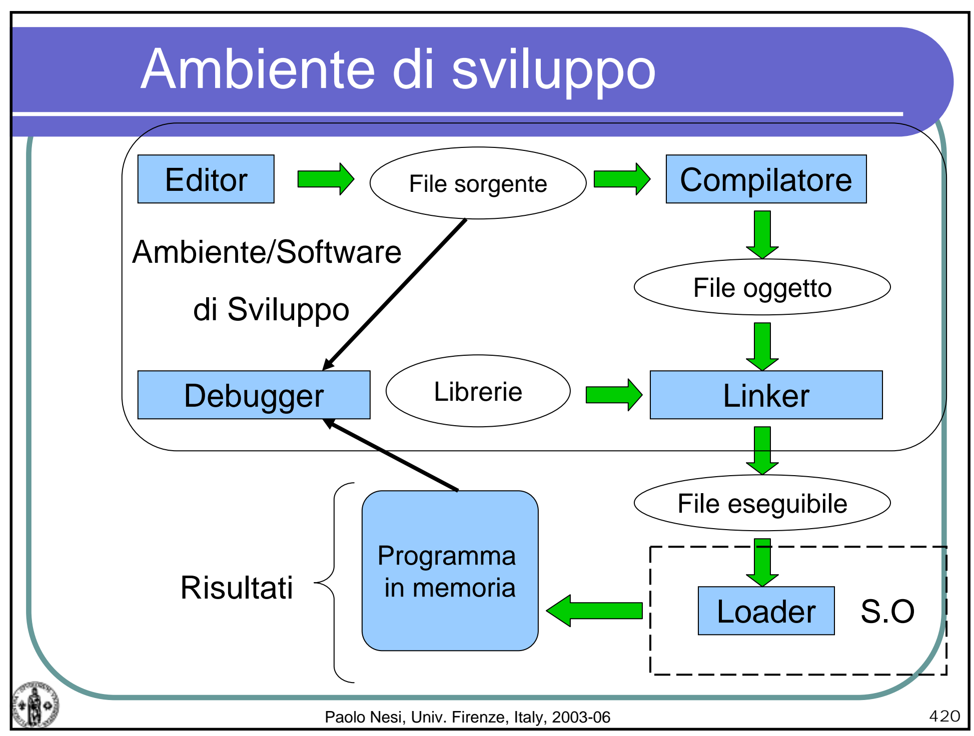
- Gli **interpreti** sono programmi capaci di eseguire direttamente un programma in linguaggio *L* istruzione per istruzione.
  - Verificano la correttezza sintattica di ogni istruzione durante l'esecuzione.
  - In presenza di istruzioni ripetute (cicli) queste sono verificate e tradotte come se fossero da eseguire per la prima volta.
- Prestazioni:
  - I programmi compilati sono in generale più efficienti e più veloci di quelli interpretati.

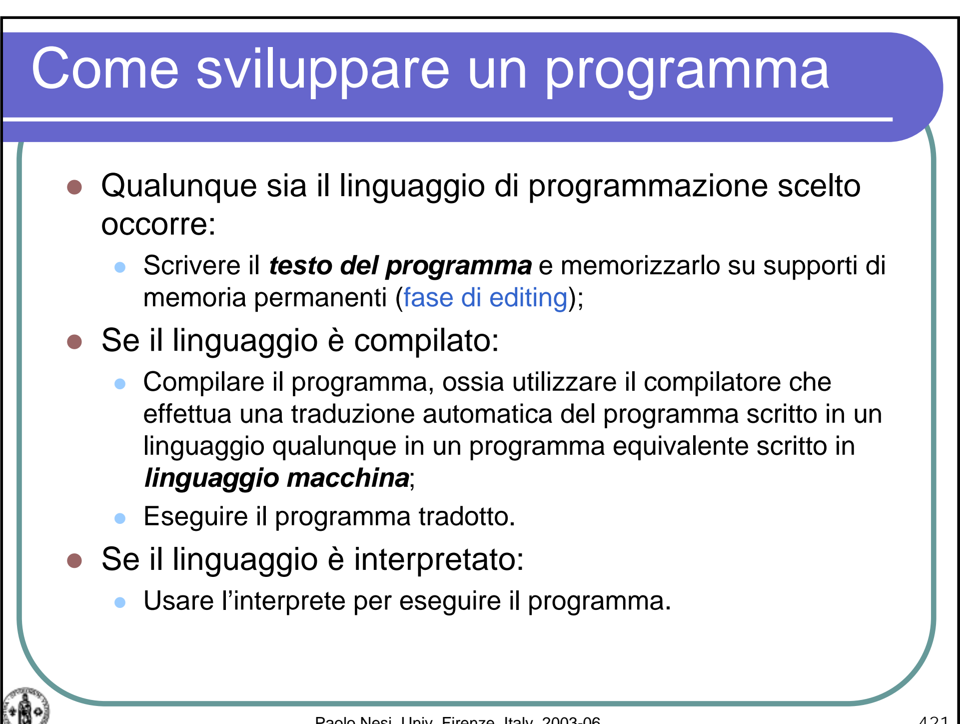
Paolo Nesi, Univ. Firenze, Italy, 2003-06 418

Le fasi della compilazione:

- **Compilatore: opera la traduzione di un programma *sorgente*** (scritto in un linguaggio ad alto livello) **in un *programma oggetto.***
- **Linker:** *(collegatore)* nel caso in cui la costruzione del programma oggetto richieda l'unione di ***più moduli o librerie*** (compilati separatamente), il linker provvede a **collegarli** formando un unico *programma eseguibile.*
- **Debugger:** consente di **eseguire passo-passo** un programma, **controllando via via quel che succede**, al fine di ***scoprire ed eliminare errori*** non rilevati in fase di compilazione.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 419


Ambiente di sviluppo
Editor
File sorgente
Compilatore
Ambiente/Software
di Sviluppo
File oggetto
Debugger
Librerie
Linker
File eseguibile
Risultati
Programma
in memoria
Loader
S.O
Paolo Nesi, Univ. Firenze, Italy, 2003-06
420


Come sviluppare un programma
Qualunque sia il linguaggio di programmazione scelto occorre:
Scrivere il testo del programma e memorizzarlo su supporti di memoria permanenti (fase di editing);
Se il linguaggio è compilato:
Compilare il programma, ossia utilizzare il compilatore che effettua una traduzione automatica del programma scritto in un linguaggio qualunque in un programma equivalente scritto in linguaggio macchina;
Eseguire il programma tradotto.
Se il linguaggio è interpretato:
Usare l'interprete per eseguire il programma.
Paolo Nesi, Univ. Firenze, Italy, 2003-06
421

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facolta' di Ingegneria,
Universita' degli Studi di Firenze

Nuovo Ordinamento

Parte 9, La Programmazione in Assembly

Dr. Ing. Ivan Bruno
Prof. Paolo Nesi
http://www.dsi.unifi.it/~ivanb
ivanb@dsi.unifi.it
AA 2005-2006

Linguaggio Assembly (ASM-86)

- 2 tipi di DICHIARAZIONI
  - ***Direttive***
    - danno informazioni all'assemblatore
    - non sono tradotte in codice macchina
  - ***Istruzioni***
    - Sono tradotte in istruzione codice macchina dall'assemblatore: 1 istruzione => 1 codice
- ***Mnemonici***
  - acronimi che indicano il tipo di istruzione
- ***Identificatori***
  - stringhe usate per rappresentare indirizzi, numeri, ….

Come è fatto un programma in Assembly

- Un programma assembly può essere diviso in 3 segmenti:
  - **Code segment** (**CS**), definisce il programma principale => l'inizio delle istruzioni.
  - **Data segment** (**DS**), definisce l'inizio dell'area dei dati che verrano usati
  - **Stack segment** (**SS**), definisce la Stack.
- I segmenti possono essere allocati in zone diverse di memoria e lontani tra loro

Definizione Segmenti

Address

CS 0100:0001

DS 1750:0001

SS 3230:0001

Indirizzi Logici

- Tutti i programmi sono costituiti da 1 o più segmenti
- Durante l'esecuzione i registri di segmento puntano ai segmenti attivi
- 64kbyte di dimensione massima

Registri segmento

- Quando MS-DOS inizia l'esecuzione del programma, inizializza 2 registri segmenti:
  - CS : code segment - il programma principale
  - SS : stack segment - lo Stack
- A questo punto la responsabilità della gestione dei registri segmento è del programmatore
- Per accedere ai dati correttamente si deve inizializzare il data segment register e se c'è anche quello extra (ES):

```
MOV  AX, DATA_SEG
MOV  DS, AX
```

Definizione dei segmenti

- Tutti i programmi sono costituiti da 1 o più segmenti
  - Code Segment; Data Segment; Stack Segment
- ogni segmento è definito così:

<Segment_Name> **segment** (<align>) (<combine>) (<class>)
<code>
<Segment Name> **ends**

Nome dei Segmenti

- La direttiva Segment richiede una label/nome (segment name)
- segment name -> indirizzo del segmento.
- E' necessario specificare il nome del segmento anche nel campo specifico della direttiva ENDS che definisce la fine del segmento
- I segmenti vengono caricati in memoria nell'ordine in cui sono scritti nel file.

Tipicamente 3 segmenti

```
CODE_SEG     SEGMENT     PARA    PUBLIC   'CODE'
             :           :
CODE_SEG     ENDS

DATA_SEG     SEGMENT     PARA    PUBLIC   'DATA'
             :           :
DATA_SEG     ENDS

STACK_SEG SEGMENT        PARA    STACK        'STACK'
          DW          1024   dup (?)
STACK_SEG    ENDS
```

Direttiva Align

- Il parametro align (allineamento) è opzionale
  - può valere: byte, word, para, or page.
  - Dice all'assemblatore, al linker, e al DOS dove caricare il segmento in memoria
  - Default : vale paragraph (16 bytes).
    - BYTE: carica il segmento in memoria iniziando al primo byte disponibile dopo l'ultimo segmento.
    - WORD: il segmento inizia al primo byte utile con indirizzo pari dopo l'ultimo segmento.
    - PARA: il segmento inizia ad un indirizzo pari divisibile per 16 (10h)
    - PAGE: il segmento inizia all'indirizzo della pagina di memoria successiva (indirizzo multiplo di 256 o 100H)

Direttiva Combine

- Definisce come il linker deve combinare i segmenti che hanno lo stesso nome, ma che compaiono in file diversi.
- **PRIVATE**: (default) non combina i segmenti di moduli diversi
- **PUBLIC o MEMORY**: concatena tutti i segmenti con lo stesso nome
- **STACK**: concatena tutti i segmenti di stack con lo stesso nome
- **COMMON**: sovrappone i segmenti con lo stesso nome
- **AT** exp: forza l'inizio del segmento all'indirizzo exp.

Direttiva Class

- Specifica di combinare i segmenti con nome di segmento diversi ma stesso class type
- Questo operando è rappresentato da un simbolo racchiuso da apostrofi
- Tipicamente si usa come class: 'CODE', 'DATA' e 'STACK'.

Direttiva Assume

Formato:

ASSUME segm_reg:segm_name {, segm_reg:segm_name}

Significato:

- Associa il registro di segmento segm_reg al segmento di nome segm_name.
- La direttiva ASSUME non provvede al caricamento dell'indirizzo del segmento nel registro relativo.

Es: **ASSUME CS:**<code segment name>**, DS:**<data segment name>**, SS:**<stack segment name>

- L'informazione fornita da ASSUME viene utilizzata dall'assemblatore per determinare il registro di segmento da utilizzare per il calcolo degli indirizzi fisici di variabili.

Assume

- Può essere usata dovunque all'interno del codice
- Eventuali direttive dello stesso tipo possono rispecificare i registri di segmento in qualunque momento del programma

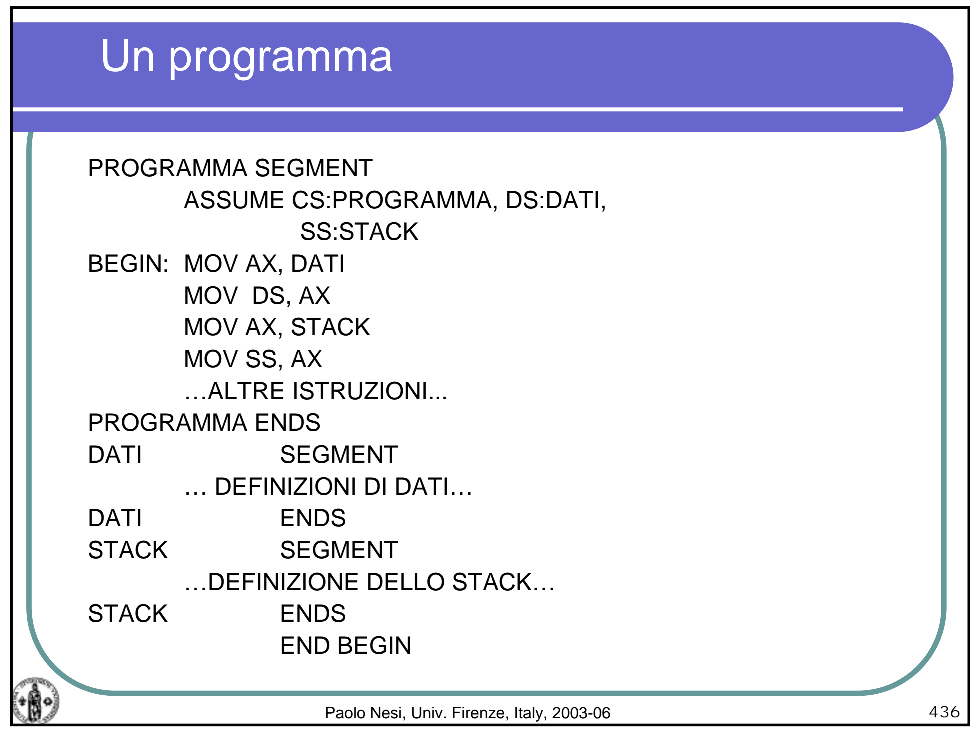
La direttiva END

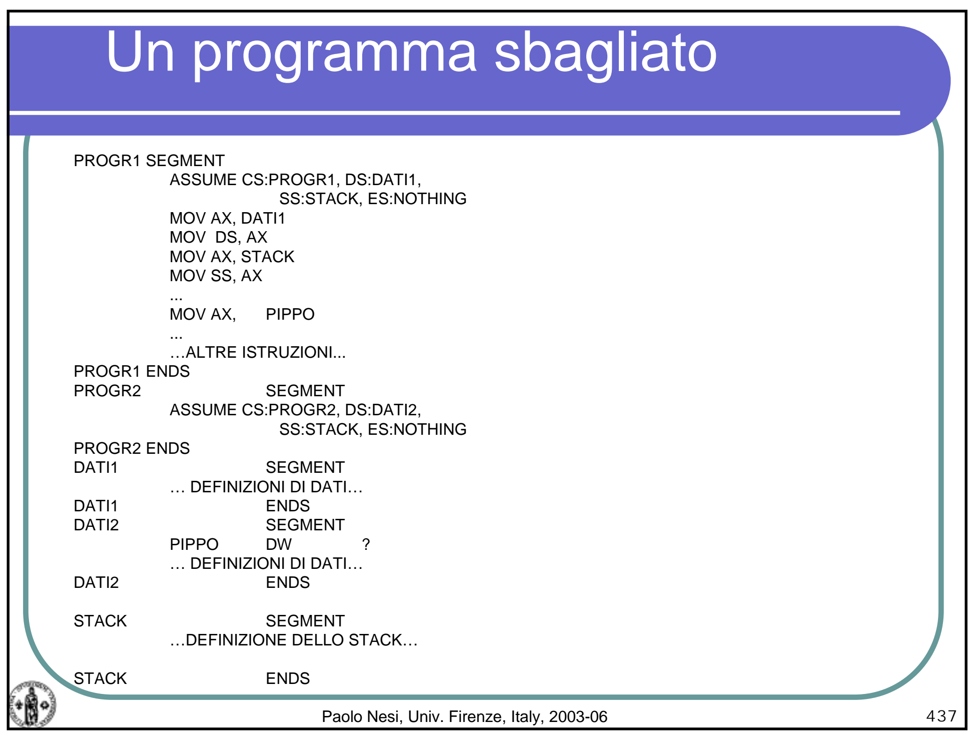
- Formato

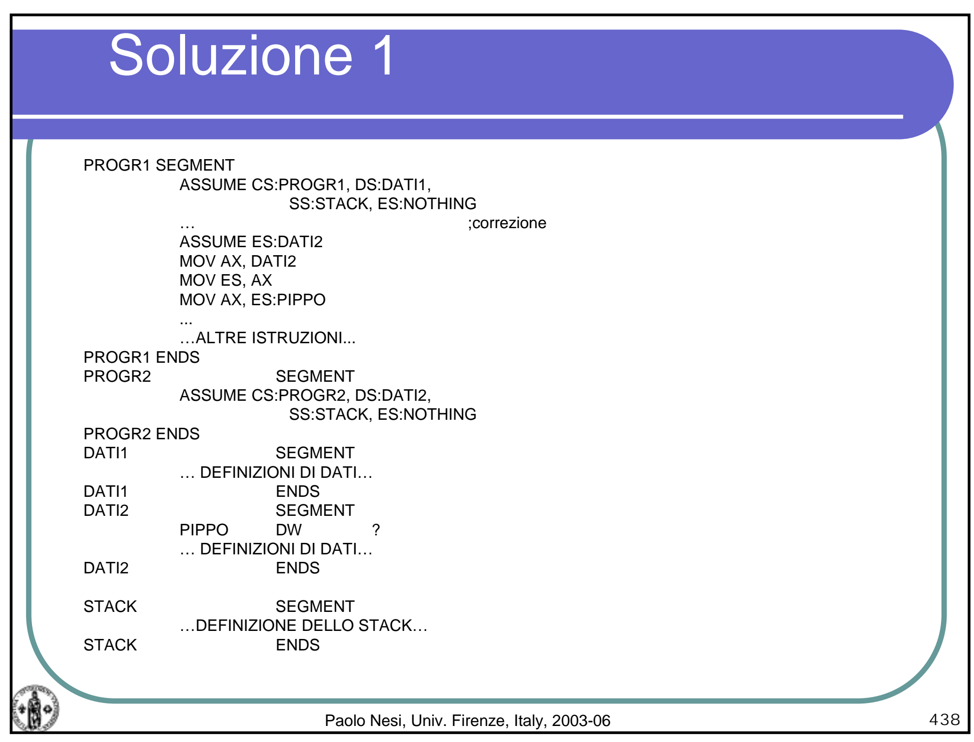
  END <LABEL>
- Significato:

  Conclude un modulo di programma.

  Se il modulo contiene un'etichetta alla prima istruzione del programma, la relativa etichetta deve essere specificata come operando. In tal modo l'assemblatore (ed il linker) possono comunicare al processore l'istruzione da cui iniziare l'esecuzione del programma.


Un programma
PROGRAMMA SEGMENT
ASSUME CS:PROGRAMMA, DS:DATI,
SS:STACK
BEGIN: MOV AX, DATI
MOV DS, AX
MOV AX, STACK
MOV SS, AX
…ALTRE ISTRUZIONI...
PROGRAMMA ENDS
DATI SEGMENT
… DEFINIZIONI DI DATI…
DATI ENDS
STACK SEGMENT
…DEFINIZIONE DELLO STACK…
STACK ENDS
END BEGIN
Paolo Nesi, Univ. Firenze, Italy, 2003-06
436


Un programma sbagliato
PROGR1 SEGMENT
ASSUME CS:PROGR1, DS:DATI1,
SS:STACK, ES:NOTHING
MOV AX, DATI1
MOV DS, AX
MOV AX, STACK
MOV SS, AX
...
MOV AX, PIPPO
...
...ALTRE ISTRUZIONI...
PROGR1 ENDS
PROGR2 SEGMENT
ASSUME CS:PROGR2, DS:DATI2,
SS:STACK, ES:NOTHING
PROGR2 ENDS
DATI1 SEGMENT
... DEFINIZIONI DI DATI…
DATI1 ENDS
DATI2 SEGMENT
PIPPO DW ?
... DEFINIZIONI DI DATI…
DATI2 ENDS
STACK SEGMENT
...DEFINIZIONE DELLO STACK…
STACK ENDS
Paolo Nesi, Univ. Firenze, Italy, 2003-06
437

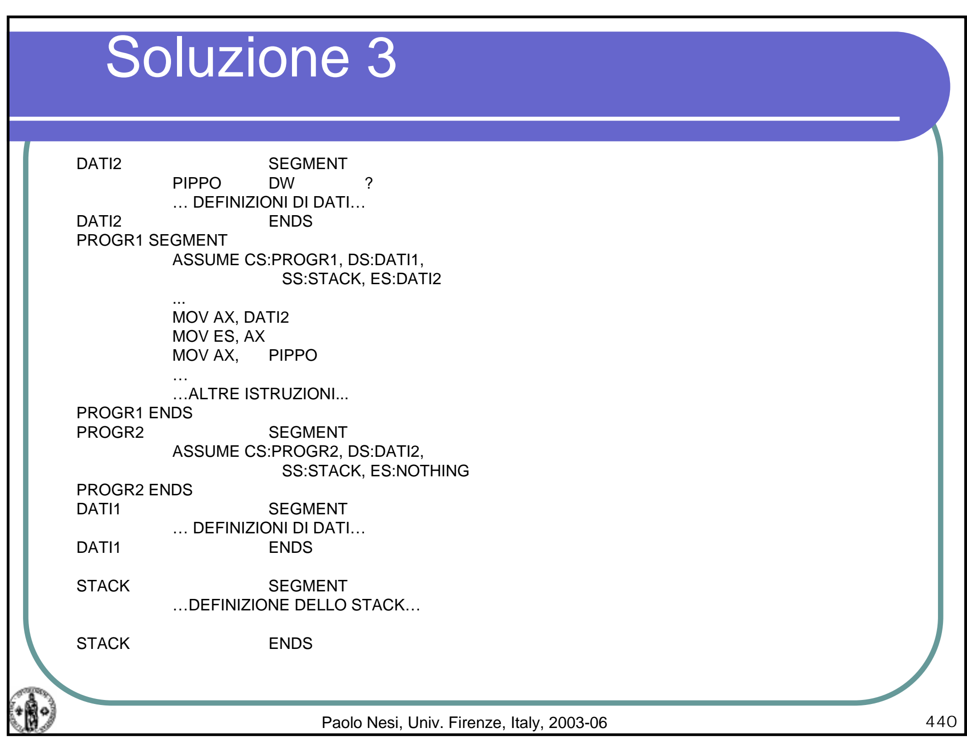


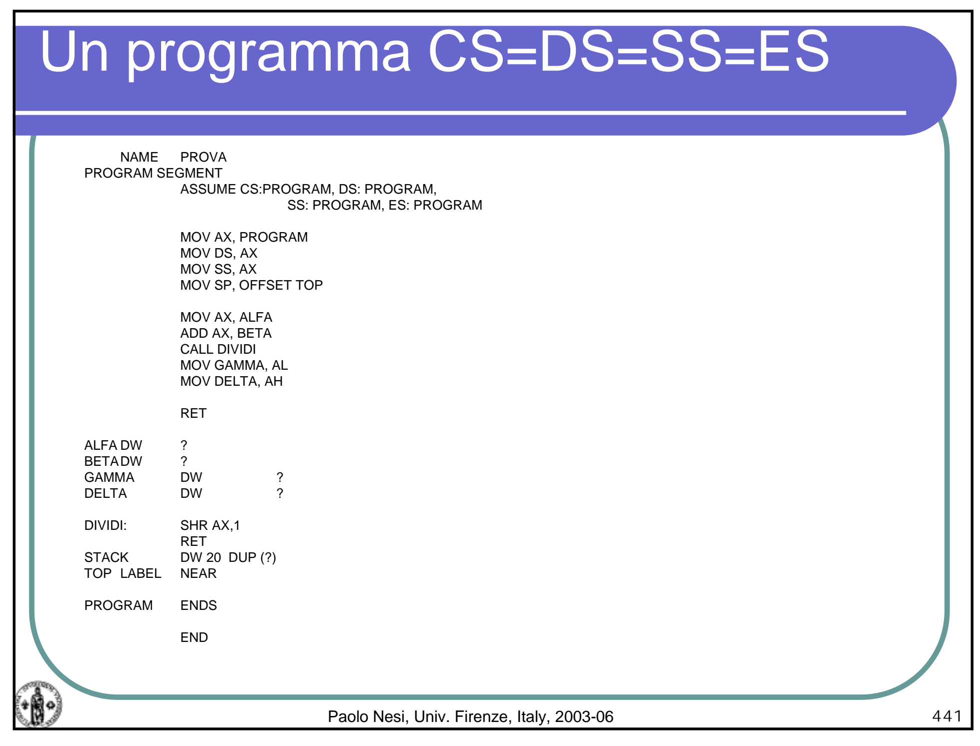
Soluzione 2

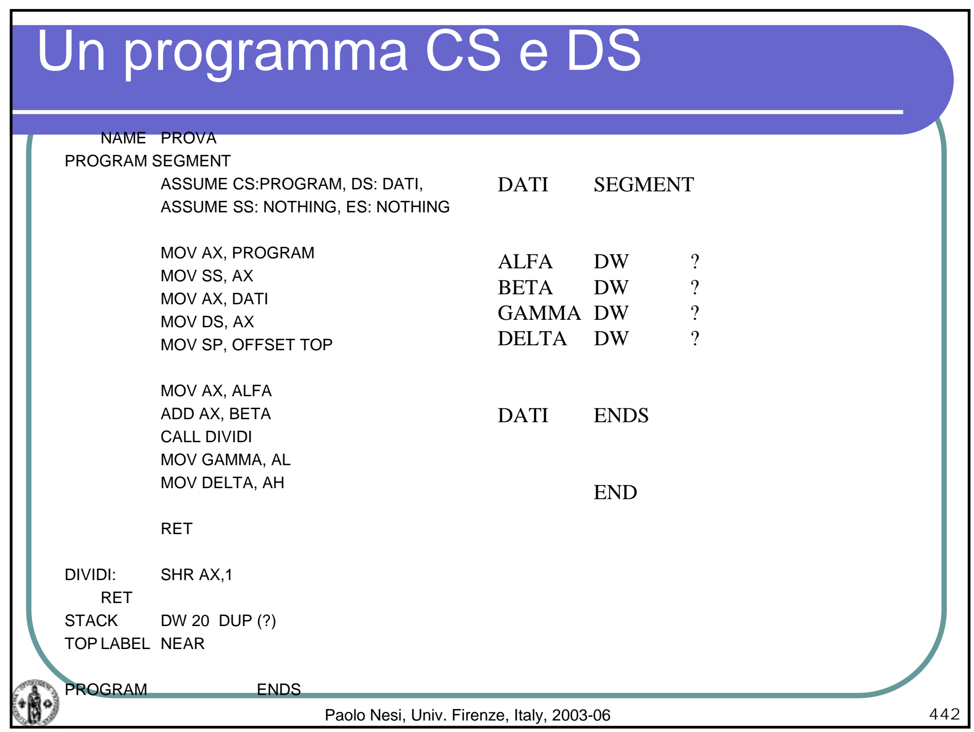
```
PROGR1 SEGMENT
        ASSUME CS:PROGR1, DS:DATI1,
               SS:STACK, ES:NOTHING
        ...                         ;correzione
        ASSUME DS:DATI2
        MOV AX, DATI2
        MOV DS, AX
        MOV AX,    PIPPO
        …                           ;eventualmente ripristinare DS a DATI1
        …ALTRE ISTRUZIONI...
PROGR1 ENDS
PROGR2          SEGMENT
        ASSUME CS:PROGR2, DS:DATI2,
               SS:STACK, ES:NOTHING
PROGR2 ENDS
DATI1           SEGMENT
        … DEFINIZIONI DI DATI…
DATI1           ENDS
DATI2           SEGMENT
        PIPPO   DW      ?
        … DEFINIZIONI DI DATI…
DATI2           ENDS

STACK           SEGMENT
        …DEFINIZIONE DELLO STACK…
STACK           ENDS
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 439


Soluzione 3
DATI2 SEGMENT
PIPPO DW ?
… DEFINIZIONI DI DATI…
DATI2 ENDS
PROGR1 SEGMENT
ASSUME CS:PROGR1, DS:DATI1,
SS:STACK, ES:DATI2
...
MOV AX, DATI2
MOV ES, AX
MOV AX, PIPPO
…
…ALTRE ISTRUZIONI...
PROGR1 ENDS
PROGR2 SEGMENT
ASSUME CS:PROGR2, DS:DATI2,
SS:STACK, ES:NOTHING
PROGR2 ENDS
DATI1 SEGMENT
… DEFINIZIONI DI DATI…
DATI1 ENDS
STACK SEGMENT
…DEFINIZIONE DELLO STACK…
STACK ENDS
Paolo Nesi, Univ. Firenze, Italy, 2003-06
440


Un programma CS=DS=SS=ES
NAME PROVA
PROGRAM SEGMENT
ASSUME CS:PROGRAM, DS: PROGRAM,
SS: PROGRAM, ES: PROGRAM
MOV AX, PROGRAM
MOV DS, AX
MOV SS, AX
MOV SP, OFFSET TOP
MOV AX, ALFA
ADD AX, BETA
CALL DIVIDI
MOV GAMMA, AL
MOV DELTA, AH
RET
ALFA DW ?
BETA DW ?
GAMMA DW ?
DELTA DW ?
DIVIDI: SHR AX,1
RET
STACK DW 20 DUP (?)
TOP LABEL NEAR
PROGRAM ENDS
END
Paolo Nesi, Univ. Firenze, Italy, 2003-06
441



Linguaggio Assembly (ASM-86)

- Formato delle istruzioni Assembly:

  Label: Mnemonico Operando, Operando ;Commento

- Label
  - è un identificatore cui è assegnato l'indirizzo del primo byte dell'istruzione alla quale si riferisce
  - Può servire come nome simbolico cui riferirsi nei salti condizionati e incondizionati
  - i due punti sono necessari se essa è presente altrimenti non ci sono

Paolo Nesi, Univ. Firenze, Italy, 2003-06 443

Linguaggio Assembly (ASM-86)

- Mnemonico
  - indica l'operazione vera e propria
    - es: operazione di somma => ADD
- Operando
  - la presenza dipende dall'istruzione
  - possiamo avere istruzioni con 1, 2 o nessun operando
- Commenti
  - consentono di commentare l'istruzione
  - possono contenere qualsiasi carattere
  - sono introdotti dal punto e virgola
  - se non ci sono, il punto e virgola può essere omesso

Linguaggio Assembly (ASM-86)

- No key-sensitive
- Variabile
  - è un identificatore (stringa) associato al primo byte di un dato, il cui valore può cambiare durante il programma. Es: ALBERO
- Espressione
  - è una concatenazione di simboli chiamati TOKENS

Linguaggio Assembly (ASM-86)

- Tokens
  - identificatori variabile
  - oppure possono essere:
    - *costante*
      - un numero con suffisso che indica la base: B o b (binario), D o d (decimale), O o o (ottale), H o h (esadecimale). Per default la base è decimale
      - In caso di base esadecimale la prima cifra a sx deve essere compreso tra 0 e 9 es: BC2H deve essere scritto 0BC2H altrimenti verrebbe confusa con una stringa
    - *costante stringa*
      - qualsiasi combinazione di caratteri fra apici. Es: ‘ALBERO’

Linguaggio Assembly (ASM-86)

- Tokens
  - *operatore aritmetico*
    - +, -, * , / , MOD, SHL, SHR
  - *operatore logico*
    - AND, OR, NOT, XOR
  - *relazionali*
    - EQ, NE, LT, GT, LE, GE
  - *operatori che ritornano un valore*
    - $, SEG, OFFSET, LENGTH, TYPE
  - *attributi*
    - PTR, DS:, ES:, SS:, CS:, HIGH, LOW

Linguaggio Assembly (ASM-86)

- Gli identificatori sono stringhe di non più di 31 caratteri
  - Il primo carattere non può essere un numero: 1ALBERO è sbagliato!!!
  - il primo carattere può essere una lettera (a-z, A-Z), oppure uno dei 4 caratteri @ _ $?
  - gli altri caratteri possono essere una lettera, un numero, o uno dei 4 caratteri sopra.
- Non possono essere:
  - nomi di registri (AX, BX, ….)
  - mnemonici (ADD, SUB,…)
  - nomi di operatori speciali

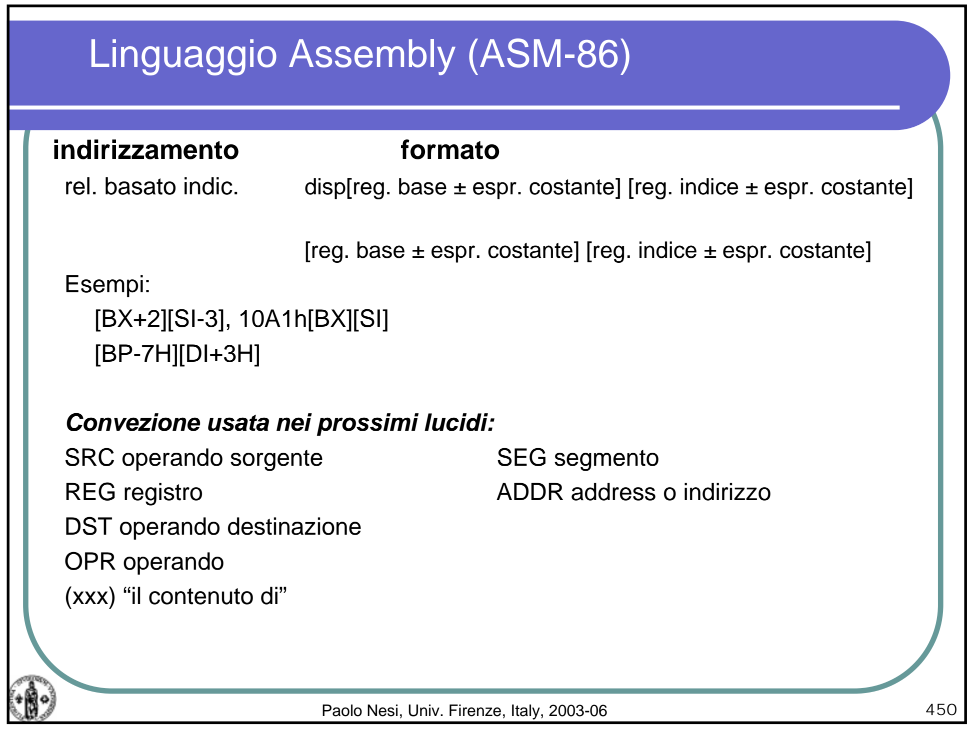
Paolo Nesi, Univ. Firenze, Italy, 2003-06 448

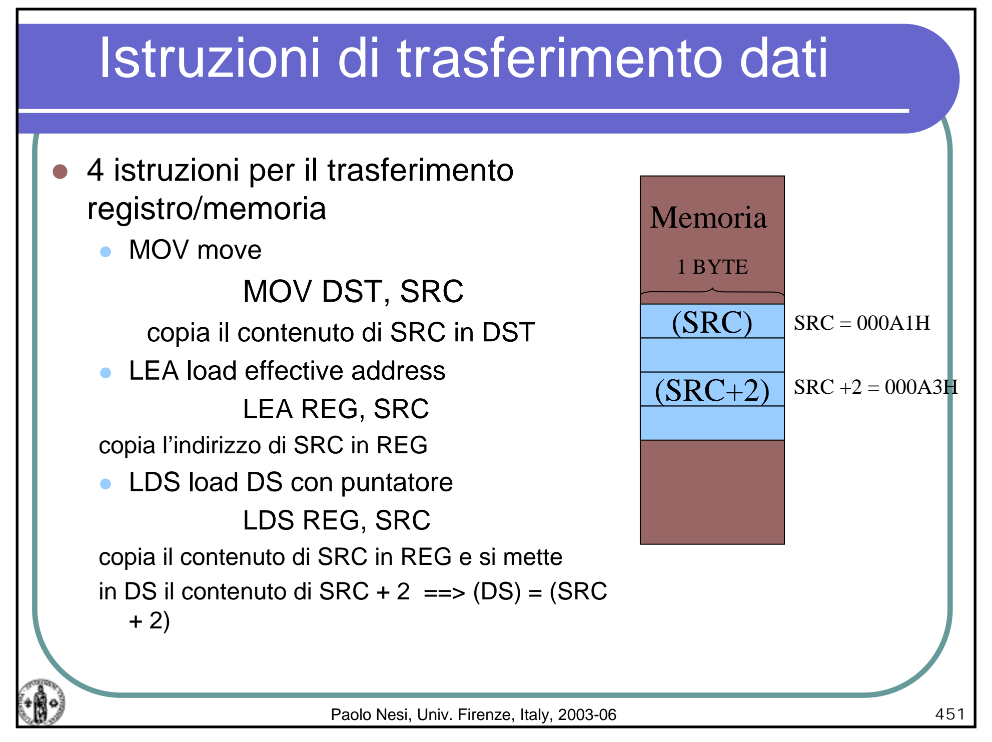
Linguaggio Assembly (ASM-86)

- Formato Operandi
  - dipende dal modo di indirizzamento
- L'8086 ha 7 modi di indirizzamento per i dati

indirizzamento	formato	esempio
immediato (dato)	espressione costante	10011B, 529, 529D, 'Acqua'
diretto (Effective Address)	nome var. ± espr. costante	CNT - 5, [10A0h], CNT
	nome var.[± espr. costante]	CNT[-5]
registro	registro	AX
registro indiretto	[registro]	[BX] (contenuto di BX)
registro relativo	disp[reg. ± espr. costante]	[AX+10H],10h[AX]
	[reg. ± espr. costante]	[BX-7D]
basato indicizzato	[reg. base][reg. indice]	[BX][SI], [BP][DI]

Paolo Nesi, Univ. Firenze, Italy, 2003-06 449


Linguaggio Assembly (ASM-86)
indirizzamento
formato
rel. basato indic.
disp[reg. base ± espr. costante] [reg. indice ± espr. costante]
[reg. base ± espr. costante] [reg. indice ± espr. costante]
Esempi:
[BX+2][SI-3], 10A1h[BX][SI]
[BP-7H][DI+3H]
Convezione usata nei prossimi lucidi:
SRC operando sorgente
REG registro
DST operando destinazione
OPR operando
(xxx) "il contenuto di"
SEG segmento
ADDR address o indirizzo
Paolo Nesi, Univ. Firenze, Italy, 2003-06
450


Istruzioni di trasferimento dati
4 istruzioni per il trasferimento registro/memoria
MOV move
MOV DST, SRC
copia il contenuto di SRC in DST
LEA load effective address
LEA REG, SRC
copia l'indirizzo di SRC in REG
LDS load DS con puntatore
LDS REG, SRC
copia il contenuto di SRC in REG e si mette in DS il contenuto di SRC + 2 ==> (DS) = (SRC + 2)
Memoria
1 BYTE
(SRC)
(SRC+2)
SRC = 000A1H
SRC +2 = 000A3H
Paolo Nesi, Univ. Firenze, Italy, 2003-06
451

Istruzioni di trasferimento dati

- LES load ES con puntatore

LES REG, SRC

copia il contenuto di SRC in REG e si mette (SRC + 2) in ES

- XCHG istruzione di scambio

XCHG OPR1, OPR2

scambia i valori contenuti in OPR1 e OPR2

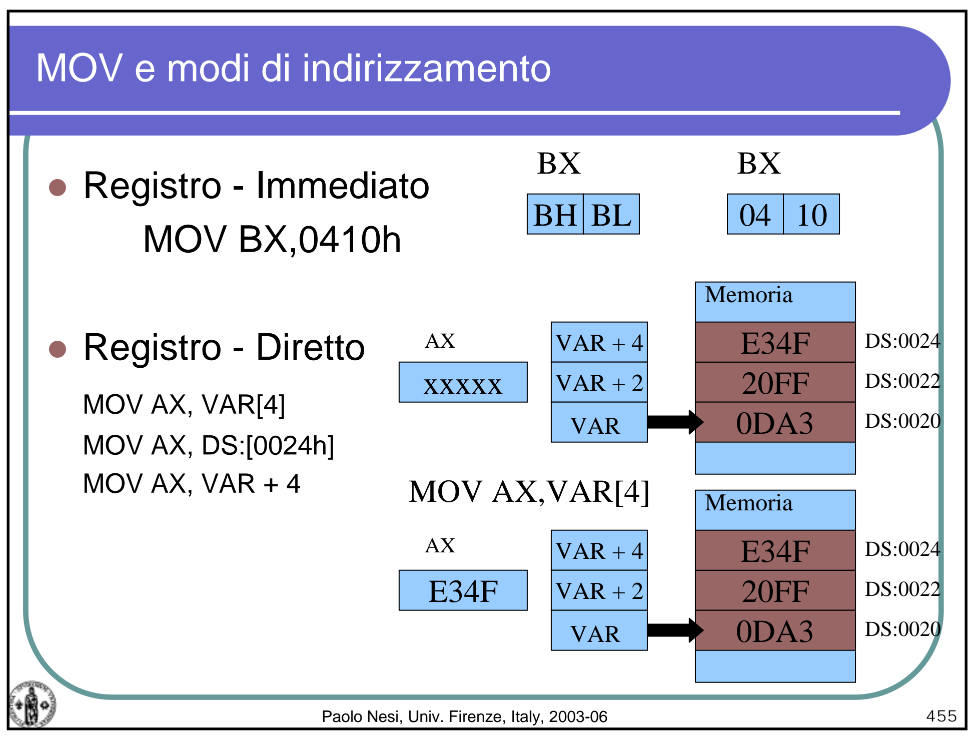
  - almeno uno dei due operandi deve essere un registro, ma nessuno dei due deve essere un registro di segmento

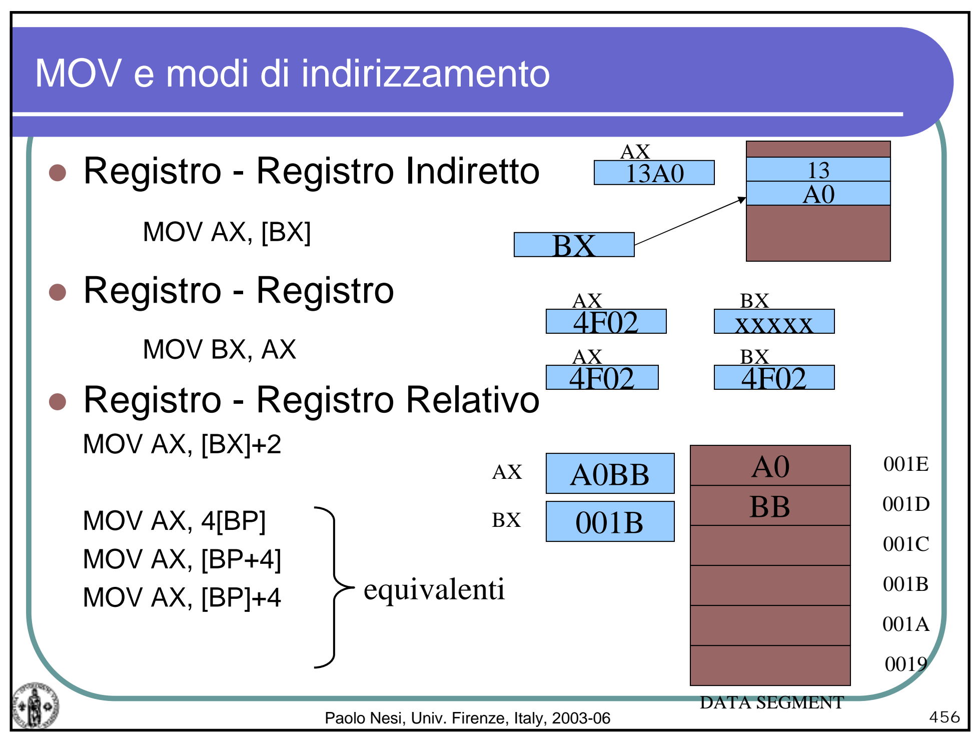
Trasferimento dati

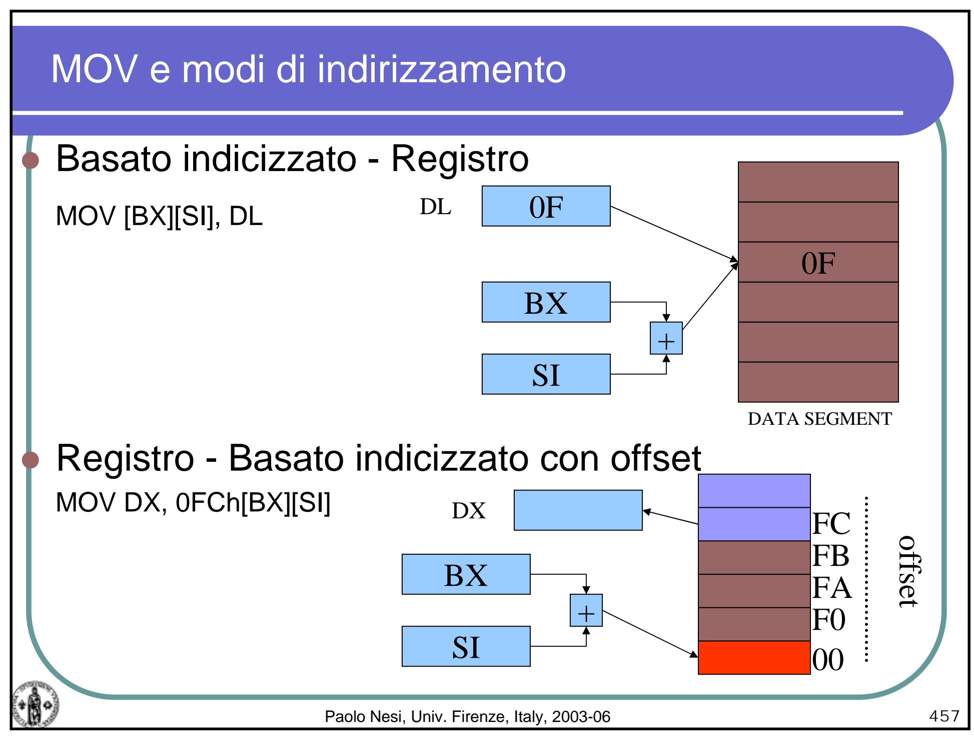
- Le istruzioni di trasferimento non affliggono i flags dell'8086
- Relativamente ai modi di indirizzamento la destinazione non può essere né un dato immediato né il registro CS
- Se la destinazione è un registro di segmento, la sorgente non può essere un dato immediato

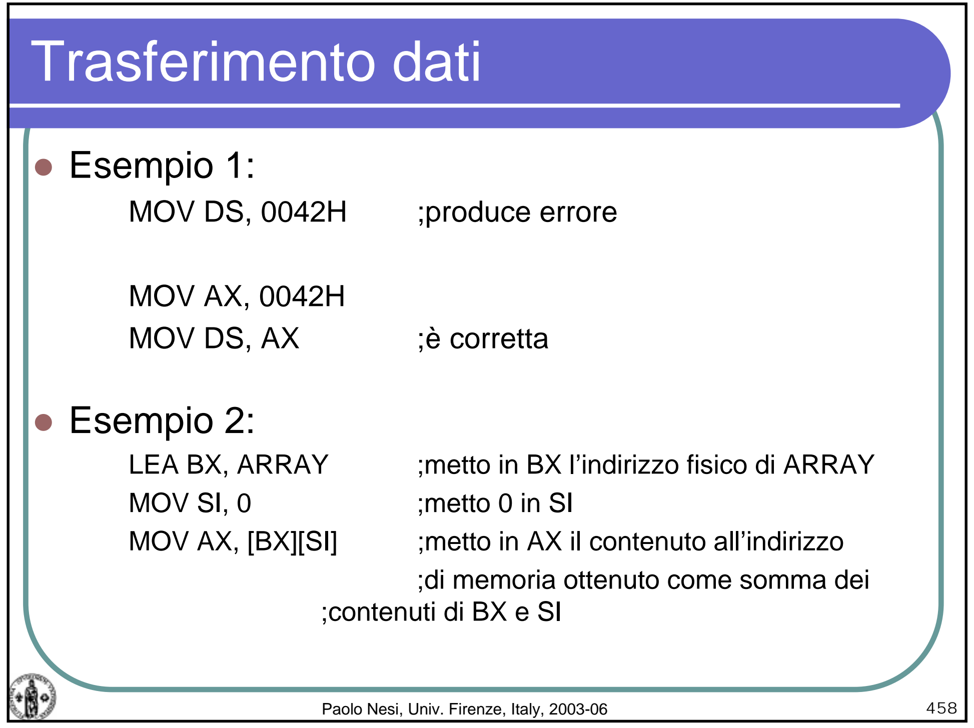
Trasferimento dati

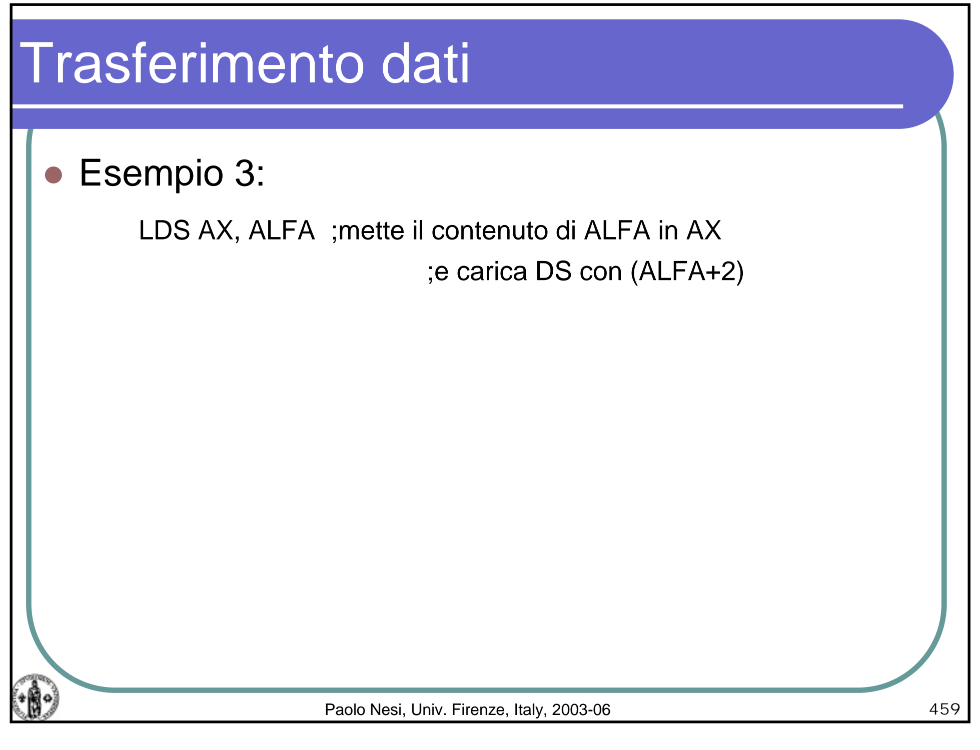
- Per LEA, LES e LDS
  - la destinazione REG non può essere un registro segmento
  - la sorgente non può avere un modo di indirizzamento immediato o registro.
- Per il MOV se uno dei due operandi è specificato col modo immediato, l'altro deve essere un registro.
- In generale: è proibito trasferire un dato immediato in un registro segmento




MOV e modi di indirizzamento
Registro - Registro Indiretto
MOV AX, [BX]
AX
13A0
13
A0
BX
Registro - Registro
MOV BX, AX
AX
4F02
BX
xxxxx
AX
4F02
BX
4F02
Registro - Registro Relativo
MOV AX, [BX]+2
AX
A0BB
BX
001B
A0
BB
001E
001D
001C
001B
001A
0019
MOV AX, 4[BP]
MOV AX, [BP+4]
MOV AX, [BP]+4
equivalenti
DATA SEGMENT
Paolo Nesi, Univ. Firenze, Italy, 2003-06
456


MOV e modi di indirizzamento
Basato indicizzato - Registro
MOV [BX][SI], DL
DL
0F
BX
SI
+
0F
DATA SEGMENT
Registro - Basato indicizzato con offset
MOV DX, 0FCh[BX][SI]
DX
BX
SI
+
FC
FB
FA
F0
00
offset
Paolo Nesi, Univ. Firenze, Italy, 2003-06
457


Trasferimento dati
Esempio 1:
MOV DS, 0042H ;produce errore
MOV AX, 0042H
MOV DS, AX ;è corretta
Esempio 2:
LEA BX, ARRAY ;metto in BX l'indirizzo fisico di ARRAY
MOV SI, 0 ;metto 0 in SI
MOV AX, [BX][SI] ;metto in AX il contenuto all'indirizzo
;di memoria ottenuto come somma dei
;contenuti di BX e SI
Paolo Nesi, Univ. Firenze, Italy, 2003-06
458

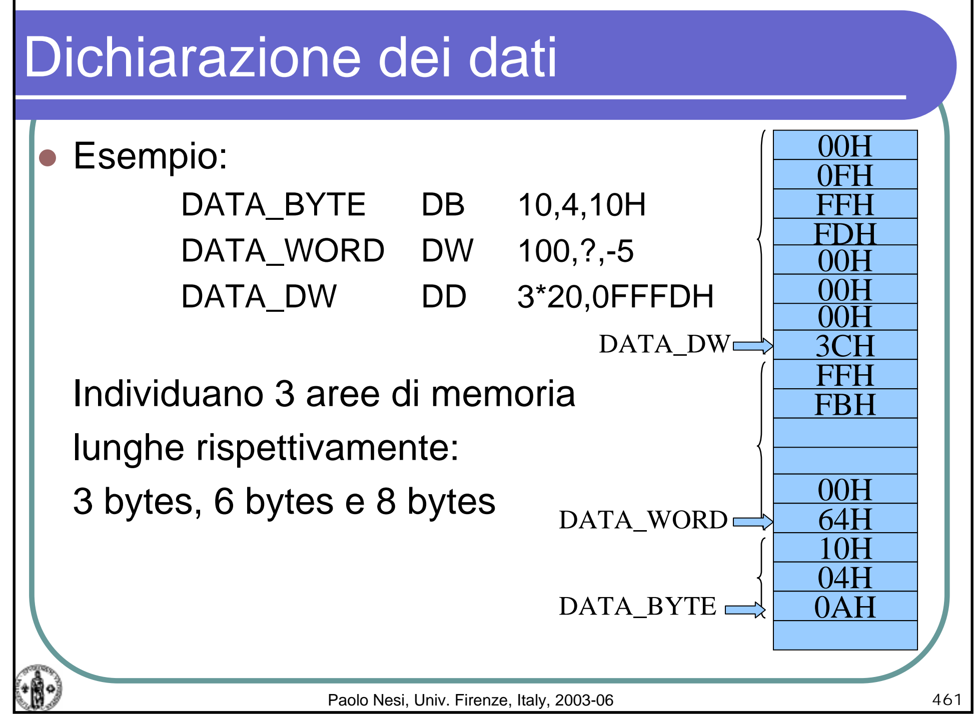

Trasferimento dati
Esempio 3:
LDS AX, ALFA ;mette il contenuto di ALFA in AX
;e carica DS con (ALFA+2)
Paolo Nesi, Univ. Firenze, Italy, 2003-06
459

Dichiarazione dei dati

- Sintassi:

  <nome var> <mnemonico> op1, op2,… ;commento
- Lo mnemonico determina la dimensione in byte
  - **DB** (define byte) ogni operando occupa 1 byte
  - **DW** (define word) ogni operando occupa 1 word = 2bytes
  - **DD** (define double word) occupa 2 word = 4bytes
- Gli operandi possono essere dei valori o espressioni costanti
- Per riservare lo spazio di memoria si usa "?"

Paolo Nesi, Univ. Firenze, Italy, 2003-06 460



Data Segment

- La prima variabile dichiarata nel Data Segment viene allocata all'indirizzo DS:0. La successiva nello spazio immediatamente dopo.

Array

- Un array è un aggregato di dati tutti dello stesso tipo
- Astrattamente un array può essre visto come un vettore
- Ciascun elemento dell'array può essere individuato mediante un indice
- Noto l'indice *i* l'indirizzo di memoria dell'elemento *i-esimo* è dato da:

  Indirizzo = Indirizzo Base + (*i* * *dimensione dato*)

Array

- Come definire un array:

<nome > **DB** | **DW**| **DD** <dim> **DUP** (<elementi>)

Dove:

- <nome > - nome della variabile array
- <dim> **DUP** (<elementi>) - duplica gli elementi specificati pari al numero di volte specificato dal valore intero *dim*

Paolo Nesi, Univ. Firenze, Italy, 2003-06 464

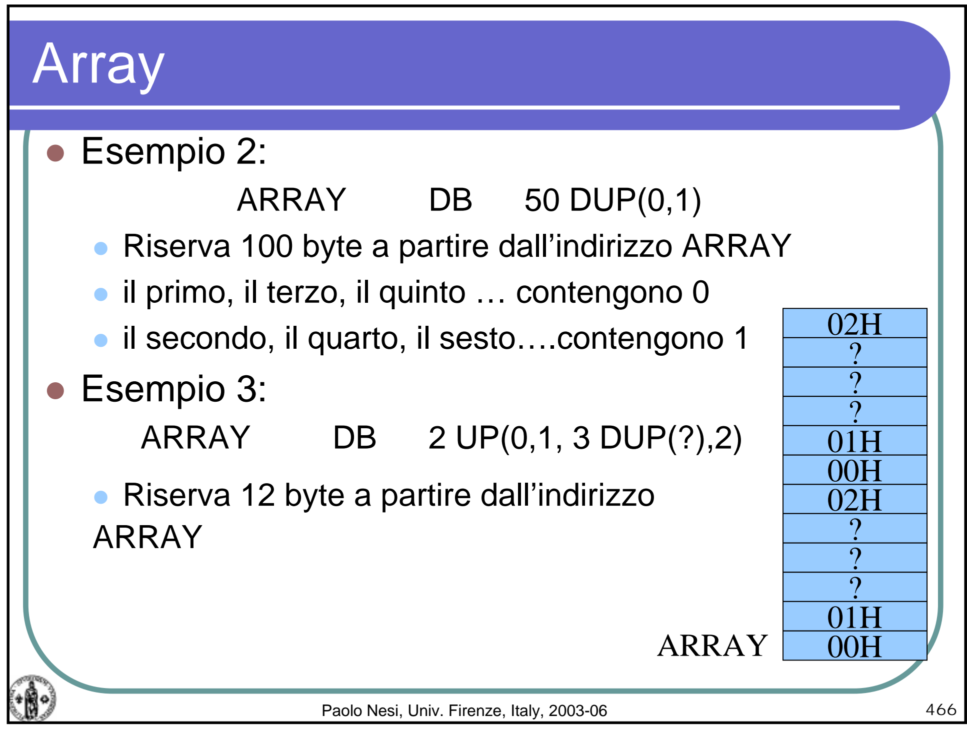
Array

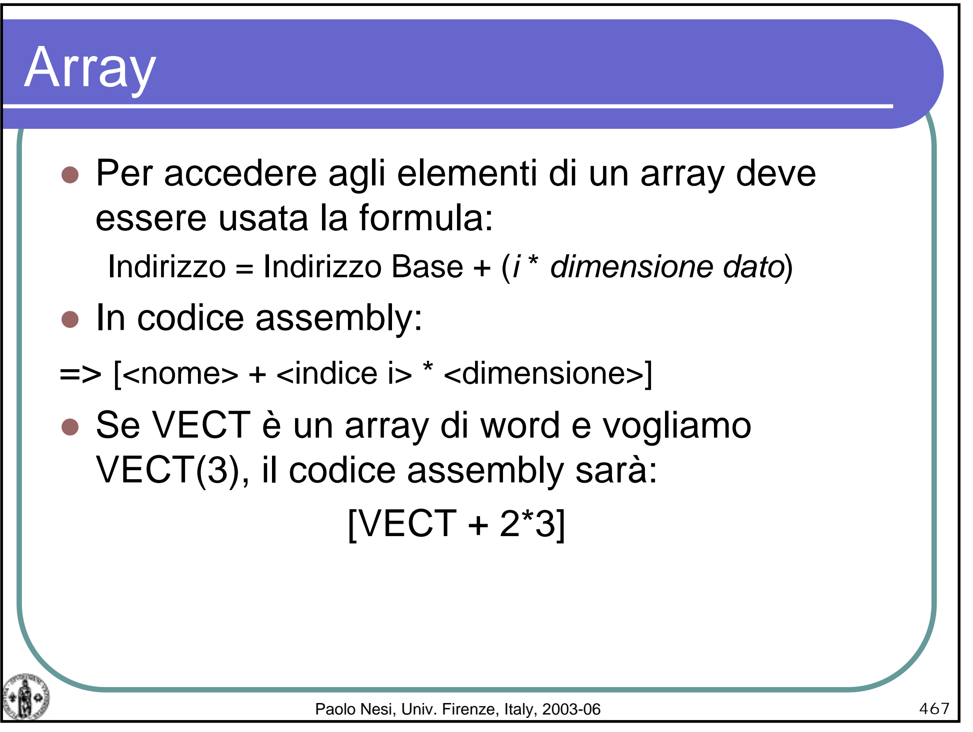
- Esempio 1:

```
V3  DB     10 DUP(?)
```

- V3 è un vettore di memoria di 10 bytes
- V3 rappresenta l'Indirizzo Base dell'array nel Data Segment

Paolo Nesi, Univ. Firenze, Italy, 2003-06 465


Array
Esempio 2:
ARRAY DB 50 DUP(0,1)
Riserva 100 byte a partire dall'indirizzo ARRAY
il primo, il terzo, il quinto … contengono 0
il secondo, il quarto, il sesto….contengono 1
Esempio 3:
ARRAY DB 2 UP(0,1, 3 DUP(?),2)
Riserva 12 byte a partire dall'indirizzo ARRAY
02H
?
?
?
01H
00H
02H
?
?
?
01H
ARRAY
00H
Paolo Nesi, Univ. Firenze, Italy, 2003-06
466


Array
Per accedere agli elementi di un array deve essere usata la formula:
Indirizzo = Indirizzo Base + (i * dimensione dato)
In codice assembly:
=> [<nome> + <indice i> * <dimensione>]
Se VECT è un array di word e vogliamo VECT(3), il codice assembly sarà:
[VECT + 2*3]
Paolo Nesi, Univ. Firenze, Italy, 2003-06
467

Stringhe

- Una stringa è una sequenza di caratteri
- In assembly ogni carattere occupa 1 byte
- Sintassi della dichiarazione:

<nome> DB ‘<stringa>’

Esempio:

MESSAGE DB ‘ALBERO’

o equivalentemente

MESSAGE DB ‘A’,’L’,’B’,’E’,’R’,’O’

Stringhe

- La variabile MESSAGE ha l’indirizzo del byte corrispondente ad ‘A’
- Si possono usare anche le direttive DW e DD, ma queste invertono l’ordine della stringa

MESSAGE DW ‘ALBERO’

nel primo byte abbiamo ‘O’, nel secondo ‘R’

Assembler e Tipi di dati

- Le direttive DB, DW e DD specificano il tipo dei dati
- Il tipo di dato è utile all'assemblatore per riconoscere eventuali errori nel codice.
- Esempio:

```
    …
    MOV OP1+1,AX
    MOV OP2,AL
    …
OP1 DB 1,2
OP2 DW 1010,2020
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 470

Assembler e Tipi di dati

- Il 2 dovrebbe essere rimpiazzato dal valore contenuto in AL mentre la parte alta AH andrebbe sul primo byte di OP2
- Ma il primo byte di OP2 verrebbe rimpiazzato dal contenuto di AL
- Conclusione:
  - OP1 è un BYTE mentre AX 2 byte
  - OP2 è una word mentre AL 1 byte

    $\Rightarrow$ l'assembler non assembla le due istruzioni e notifica un messaggio di errore

Paolo Nesi, Univ. Firenze, Italy, 2003-06 471

Istruzioni Aritmetiche

- Nell'8086 si possono realizzare operazioni aritmetiche (tra numeri interi) in:
  - BINARIO
    - con segno
    - senza segno
  - BCD
    - a pacchetti
    - non a pacchetti
- Si possono eseguire tutte e 4 le operazioni +,-,*,/
- Nel BCD solo l'addizione e la sottrazione

Paolo Nesi, Univ. Firenze, Italy, 2003-06 472

Istruzioni Aritmetiche

- **BCD (Binary Coded Decimal)**
- A pacchetti:
  - Sistema di numerazione che usa la base 2, che rappresenta ogni cifra decimale con 4 bit binari, utilizzando i pesi 8, 4, 2 e 1 a partire da sinistra verso destra.
  - Due cifre decimali per byte
  - Ciascun nibble di 4 bits rappresenta gli interi da 0 a 9
- Esempio:

  0011 1000

  in BCD corrisponde a 38 decimale.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 473

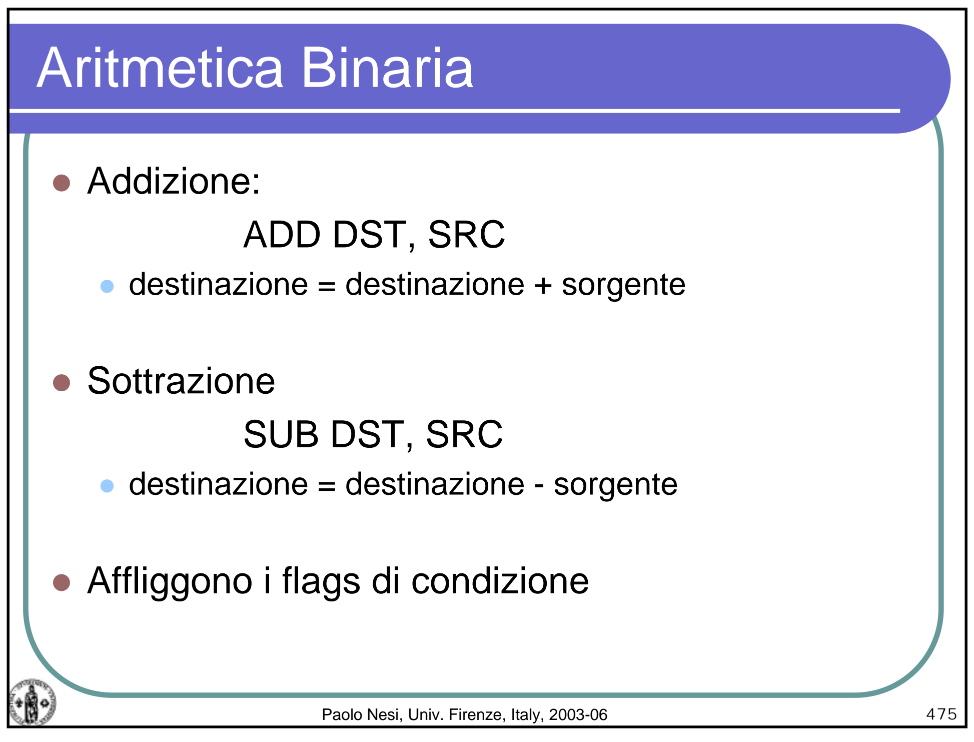
Istruzioni Aritmetiche

- **BCD (Binary Coded Decimal)**
- Non a pacchetti:
  - Una sola cifra decimale per byte
- Esempio:

  0000 1000 | 0000 0111

  in BCD corrisponde a 87 decimale.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 474



Aritmetica Binaria

- Forma generale:
  ADD {<register> | <memory>} , {<register> | <memory> | <immediate>}
  SUB {<register> | <memory>} , {<register> | <memory> | <immediate>}

- Gli operandi non possono essere entrambi riferiti in memoria, almeno uno deve essere in un registro.
  - Per sommare due elementi in memoria uno viene caricato nel registro e poi si esegue la somma registro-memoria

Paolo Nesi, Univ. Firenze, Italy, 2003-06 476

Aritmetica Binaria

- Esempio X+Y+24-Z:

```
        MOV AX, X     ;metto X in AX
        ADD AX, Y     ;aggiungo Y al contenuto di AX
        ADD AX, 24    ;ci sommo 24
        SUB AX, Z     ;ci sottraggo Z
        MOV W, AX     ;memorizzo in W il contenuto di AX
        …
X       DW    34
Y       DW    16
Z       DW    20
W       DW    ?
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 477

Aritmetica Binaria

- Poiché i registri sono a 16 bit (1 word) si avranno a disposizione solo:
  2^{16} = 64K numeri (SINGOLA PRECISIONE)
- Se si lavora a gruppi di 2 word = 32 bit (DOPPIA PRECISIONE) si possono produrre:
  $2^{32} = 2^2(2^{10})^3 = 4\ (K)^3$ = 4 miliardi di numeri
- In DOPPIA PRECISIONE occorre usare riporti e prestiti (CARRIES)

Aritmetica Binaria

- Addizione con riporto:
  ADC DST, SRC
  - destinazione = destinazione + sorgente+Carry
- Sottrazione con prestito
  SBB DST, SRC
  - destinazione = destinazione - sorgente - Carry
- Affliggono i flags di condizione

Aritmetica Binaria

- Forma generale:
  ADD {<register> | <memory>} , {<register> | <memory> | <immediate>}
  SUB {<register> | <memory>} , {<register> | <memory> | <immediate>}
- Se il bit di carry è 0 ADC si comporta come l'istruzione ADD.
- Stesso discorso per SUB e SBB
  - La sottrazione non è commutativa.

Aritmetica Binaria

Esempio 1:

```
    0123  BC62
+   0012  553A
    ----------
    0136  119C
```

```
MOV     AX, W12         ; si sposta la word più a destra
ADD     AX, W22
MOV     W32, AX
MOV     AX, W11         ; si sposta la word più a sinistra
ADC     AX, W21         ; e si somma con carry
MOV     W31, AX         ; in W31 si mette il risultato
;*****  data definition *****
W11     DW      0123h
W12     DW      BC62h
W21     DW      0012h
W22     DW      553Ah
W31     DW      ?
W32     DW      ?       ; non inizializzato
```

Aritmetica Binaria

- Esempio 2:

W = X+Y+24-Z

```
MOV AX, X        ;metto in AX la word bassa di X
MOV DX, X+2      ;metto in DX la word alta di X
ADD AX, Y        ;sommo la word bassa di Y
ADC DX, Y+2      ;sommo con riporto la word alta di Y
ADD AX, 24       ;sommo 24 alla word bassa
ADC DX, 0        ;si aggiunge alla word alta un eventuale riporto
SUB AX, Z        ;sottraggo la word bassa di Z
SBB DX, Z+2      ;sottraggo la word alta di Z ed un eventuale prestito
MOV W, AX        ;trasferisco la word bassa in W
MOV W+2, DX      ;trasferisco la word alta in W
```


Aritmetica Binaria

- Le funzioni ADD e SUB settano:
  - il flag O (overflow) in presenza di un overflow o underflow con segno
  - il flag di segno N se il risultato è negativo
  - il flag di zero Z se il risultato è zero
  - il flag di carry C in presenza di un overflow senza segno

Aritmetica Binaria

- Moltiplicazione senza segno:

  MUL SRC

  - SRC occupa 1 byte
    - (AX)=(AL)*(SRC)
  - SRC occupa 2 bytes (word)
    - (DX:AX) = (AX)*(SRC)
- Risultato senza segno
- Occorre caricare prima il registro AX con l'altro fattore
- In DX la parte alta e AX quella bassa

Aritmetica Binaria

- Moltiplicazione con segno (complemento a 2):

  IMUL SRC

  - SRC occupa 1 byte
    - (AX)=(AL)*(SRC)
  - SRC occupa 2 bytes (word)
    - (DX:AX) = (AX)*(SRC)
- Risultato con segno
- Occorre caricare prima il registro AX con l'altro fattore
- In DX la parte alta e AX quella bassa

Aritmetica Binaria

- Divisione senza segno:

  DIV SRC (src = divisore)

  - SRC occupa 1 byte
    - (AL) = quoziente di (AX)/(SRC)
    - (AH) = resto di (AX)/(SRC)
  - SRC occupa 2 bytes (word)
    - (AX) = quoziente di (DX:AX)/(SRC)
    - (DX) = resto di (DX:AX)/(SRC)
- risultato senza segno
- Occorre caricare prima il registro AX con la parte bassa e DX con quella alta

Paolo Nesi, Univ. Firenze, Italy, 2003-06 486

Aritmetica Binaria

- Divisione con segno:

  IDIV SRC

  - SRC occupa 1 byte
    - (AL) = quoziente di (AX)/(SRC)
    - (AH) = resto di (AX)/(SRC)
  - SRC occupa 2 bytes (word)
    - (AX) = quoziente di (DX:AX)/(SRC)
    - (DX) = resto di (DX:AX)/(SRC)
- risultato con segno
- Occorre caricare prima il registro AX e DX come per DIV

Paolo Nesi, Univ. Firenze, Italy, 2003-06 487

MUL & IMUL

- Esempio: differenza tra IMUL e MUL:
  - MUL B2 considera 80h come +128 mentre IMUL considera 80h come -128 (complemento a 2).

    $128 = (1000\ 0000)_2$
  - Dopo l'esecuzione di MUL 128×64 = 8192 in HEX 2000h

    quindi AX = 2000.
  - Dopo l'esecuzione di IMUL -128×64 = -8192 in HEX E000h,

    quindi AX = E000.

```
MOV     AL, B1
MUL     B2
IMUL    B2
:       :
;***** data definition *****
B1      DB      80h
B2      DB      40h
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 488

DIV e IDIV

- Se il dividendo e il divisore hanno lo stesso segno DIV e IDIV generano lo stesso risultato.
- Se sono differenti in segno,
  - DIV genera un quoziente positivo
  - IDIV genera un quoziente negativo.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 489

Conversioni di tipo

- CBW
  - converte un byte in una word
  - (AH) = estensione segno di (AL)
- CWD
  - converte una word in una double word
  - (DX) = estensione segno di (AX)
- Nessun operando aggiuntivo
- L'estensione del segno di un numero in complemento a 2 si ottiene prendendo tutti i bit del byte alto pari al valore del bit più significativo del byte basso (bit di segno).
- Es: a 16 bit 10111001 diventa 11111111 10111001
- Nessun flag viene settato

Paolo Nesi, Univ. Firenze, Italy, 2003-06 490

Conversioni di tipo

- Addizione tra un dato X in singola precisione e uno Y in doppia

```
MOV AX, X        ;metto X in AX
CWD              ;metto in DX, X con segno esteso
ADD AX, Y ;sommo la word bassa di Y ad AX
ADC DX, Y+2      ;sommo la word alta di Y a DX
MOV W, AX        ;metto la word bassa in W (parte bassa)
MOV W+2, DX      ;metto la word alta in W+2 (parte alta)
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 491

Conversioni di tipo

- Prodotto tra un byte e una word

```
BETA            DB      55H
GAMMA           DW      5555H
…
MOV AL, BETA            ;1 byte in AL
CBW                     ;estensione segno
MUL GAMMA               ;prodotto tra BETA esteso su word e
                        ;GAMMA con risultato in DX, AX
```


Conversioni di tipo

- Divisione tra una word (AX) e un byte operando

```
ALFA            DW      2222H
…
MOV AX, ALFA            ;dividendo
CBW                     ;estensione segno
DIV DH                  ;DH è il divisore
                        ;quoziente in AL e resto in AH
```

- Non è necessaria la conversione di tipo

Conversioni di tipo

- Divisione tra un byte (AL) e un altro byte operando

```
ALFA            DB    100
GAMMA           DB    7
…
MOV AL, ALFA          ;dividendo a 8 bit
CBW                   ;estensione segno a 16 bit
DIV GAMMA             ;GAMMA è il divisore
                      ;quoziente in AL e resto in AH
```

- Serve la conversione perché si usa AX
- L'estensione permette di sistemare AH

Paolo Nesi, Univ. Firenze, Italy, 2003-06 494

Conversioni di tipo

- Divisione tra un double word (DX e AX) e una word operando

```
DIVISORE            DW    4444H
…
MOV AX, "parte bassa del dividendo"
MOV DX, "parte alta del dividendo"
DIV DIVISORE ;quoziente in AX e resto in DX
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 495

Conversioni di tipo

- Divisione tra una word e una word operando

```
WORD1           DW    4444H
WORD2           DW    33
…
MOV AX, WORD1   ;dividendo a 16 bit
CWD                   ;estensione a 32 bit
DIV WORD2       ;quoziente in AX e resto in DX
```

- l'operando è una word e pertanto il divisore deve essere una double word

Paolo Nesi, Univ. Firenze, Italy, 2003-06 496

Altri operatori aritmetici

- NEG OPR esegue la negazione in complemento a 2.
  - NEG {<memory> | <register>}
- INC OPR incrementa il contenuto di 1 => OPR=OPR+1
- DEC OPR decrementa il contenuto di 1 => OPR=OPR-1
  - INC {<memory> | <register>}
  - DEC {<memory> | <register>}
- CMP OP1, OP2 comparazione tra due dati
  - CMP {<memory> | <register>}, {<memory> | <register>}
  - La comparazione avviene facendo la differenza tra i due operandi, come SUB ma non memorizza niente
- INC, DEC e NEG alterano i flags: O, S, Z e P
- NEG altera anche C
- CMP altera oppurtanamente i flags Z, C, P, S e O

Paolo Nesi, Univ. Firenze, Italy, 2003-06 497

Funzioni Logiche

- AND {destination}, {source}
  - destination = destination AND source
- OR {destination}, {source}
  - destination = destination OR source
- XOR {destination}, {source}
  - destination = destination XOR source
- NOT {destination}
  - destination = NOT destination
- TEST {destination}, {source}
  - esegue l'AND tra gli operandi ma non memorizza il risultato
- Per ciascuno:

  AND {register | memory} , {register | memory | immediate}
  OR {register | memory} , {register | memory | immediate}
  XOR {register | memory} , {register | memory | immediate}
  NOT {register | memory}

Paolo Nesi, Univ. Firenze, Italy, 2003-06 498

Funzioni Logiche e flags

- L'operando del NOT non può essere un dato immediato
- Il NOT non affligge flags
- Per OR, AND, XOR e TEST, destinazione e sorgente devono avere la stessa dimensione.
- Ad eccezione del NOT, tutte le altre istruzioni puliscono il flag di carry e overflow, e settano flag di segno col valore del bit più significativo del risultato
- Tutte le funzioni logiche lavorano bit a bit
- Sono utili nella realizzazione di maschere
- La funzione TEST è spesso usata per realizzare condizioni di confronto e per agire sui flags di condizione.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 499

Funzioni Logiche

Esempio:
Se (AL) = 1100 0101 (C5h) e (BH) = 0101 1100 (5Ch)
AND AL, BH
(AL) diventa 0100 0100 (44h)

Esempio: Mascheratura
(AX) = 0101 1010
maschera DATO => 1001 0001 setta il 1°, il 5° e l'8° bit

OR AX, DATO ==> (AX)=**1**10**1** 101**1**

Paolo Nesi, Univ. Firenze, Italy, 2003-06 500

Funzioni Logiche

Esempio :
(AX) = 0101 1010
DATO => 1001 0001

XOR AX, DATO ==> (AX) = **1**10**0** 101**1**

Esempio azzeramento bit con operazione AND:
(AX) = 0101 1010
DATO => 1001 0001

AND AX, DATO ==> (AX) = 000**1** 0000

Paolo Nesi, Univ. Firenze, Italy, 2003-06 501

Funzioni Logiche

Esempio:

(AX) = 0101 1010

DATO => 1001 0001

TEST AX, DATO ==> (AX) = 000**1** 0000

- Nel caso in cui il risultato fosse stato 0000 0000, cioè nullo, il flag Z sarebbe stato settato a 1
- Un solo 1 basta per settare il flag Z a 0.

Istruzioni di SHIFT

- Shift logico a sinistra

  SHL OPR, CNT ;shift a sinistra di (CNT)

  - Inserisce tanti zeri a partire dal bit meno significativo quanti indicati in CNT
  - I bit via via shiftati che escono dal bit più significativo vanno a finire in successione nel flag C
- Shift logico a destra

  SHR OPR, CNT ;shift a destra di (CNT)

  - Inserisce tanti zeri a partire dal bit più significativo quanti indicati in CNT
  - I bit via via shiftati che escono dal bit più basso vanno a finire in successione nel flag C

Istruzioni di SHIFT

- Shift aritmetico a sinistra

  SAL OPR, CNT ;shift a sinistra di (CNT)

  - Uguale a SHL

- Shift aritmentico a destra (conservazione del segno)

  SAR OPR, CNT ;shift a destra di (CNT)

  - Inserisce tanti bit con valore pari al segno dell'operando a partire dal bit più significativo quanti indicati in CNT
  - I bit via via shiftati che escono dal bit più basso vanno a finire in successione nel flag C

Paolo Nesi, Univ. Firenze, Italy, 2003-06

504

Istruzioni di SHIFT

- Il numero di posizioni dello shift è specificato dal contatore CNT che:
  - se è specificato in modo immediato vale 1
  - altrimenti è il contenuto del registro CL
- I flags, C, P e Z sono affetti da queste operazioni
- Shiftare a sx di n bit significa moltiplicare per 2^n
- Shiftare a dx di n bit significa dividere per 2^n
- Il flag O potrebbe essere affetto solo se l'operando sorgente è 1 e nel caso di shift a sx (si tratta di una moltiplicazione per 2 e può generare overflow).
- Il caso in cui si ha il settaggio di O è quando il nuovo bit più significativo è diverso da quello precedente e depositato nel flag C

Paolo Nesi, Univ. Firenze, Italy, 2003-06

505

Istruzioni di SHIFT

- Es 1:
  - (AH) = 1000 0000 numero negativo in complemento a 2 pari -128
  - SAL AH, 1 ==> (AH) = 0000 0000 numero positivo e C=1
  - Da un numero negativo siamo passati ad un numero positivo in seguito ad una moltiplicazione per 2. Questo viene interpretato come un overflow => O=1 e in effetti lo è, visto che -128*2=-256 fuori dinamica per una rappresentazione a 8 bit
- Es 2:
  - (AH) = 1100 0000 numero negativo in complemento a 2 pari -64
  - SAL AX, 1 ==> (AX) = 1000 0000 e C=1
  - in più 1000 0000 vale -128 il doppio di -64
  - In questo caso il risultato è giusto e non si ha overflow

Istruzioni di SHIFT

Esempio:

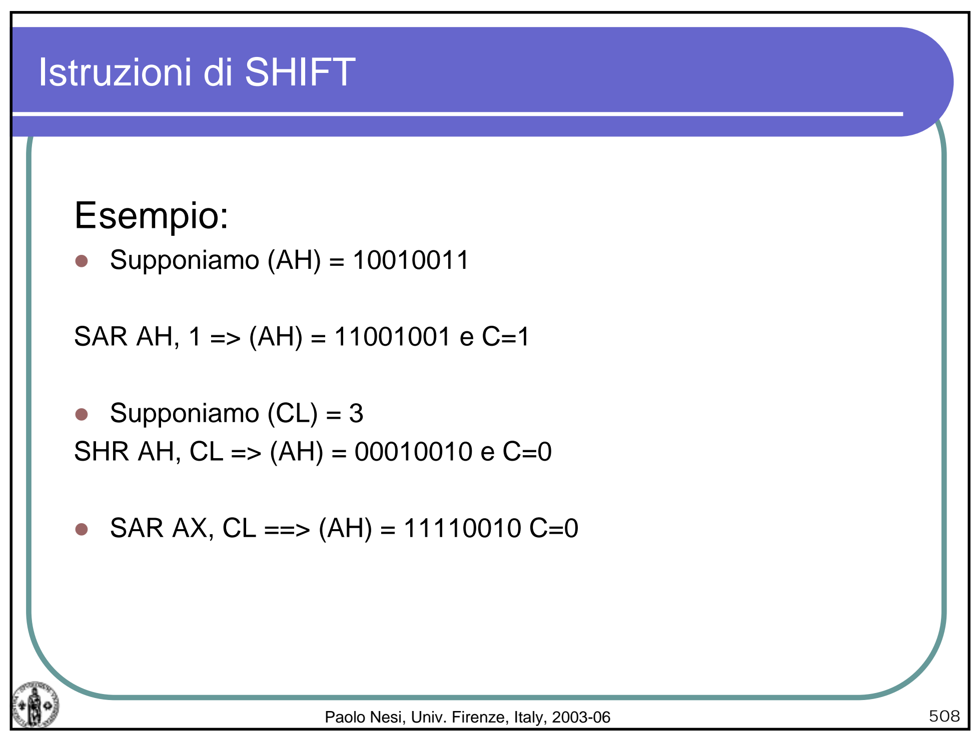
- Supponiamo (AH) = 10010011

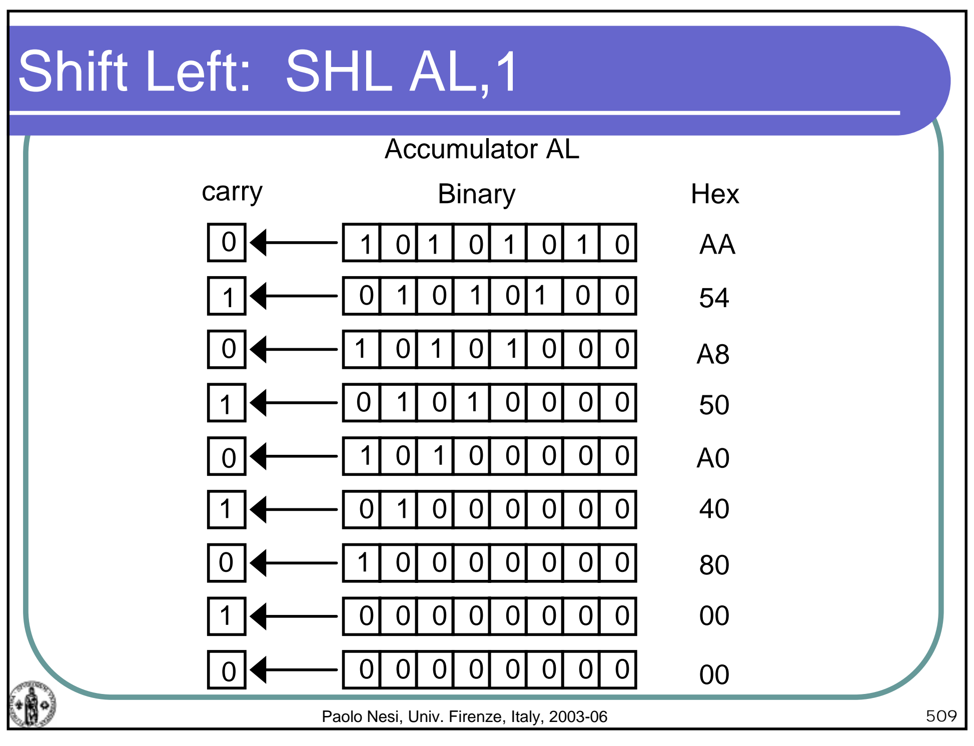
SHL AH, 1 oppure SAL AH, 1 => (AH) = 00100110 e C=1
ma per SAL il flag O = ?

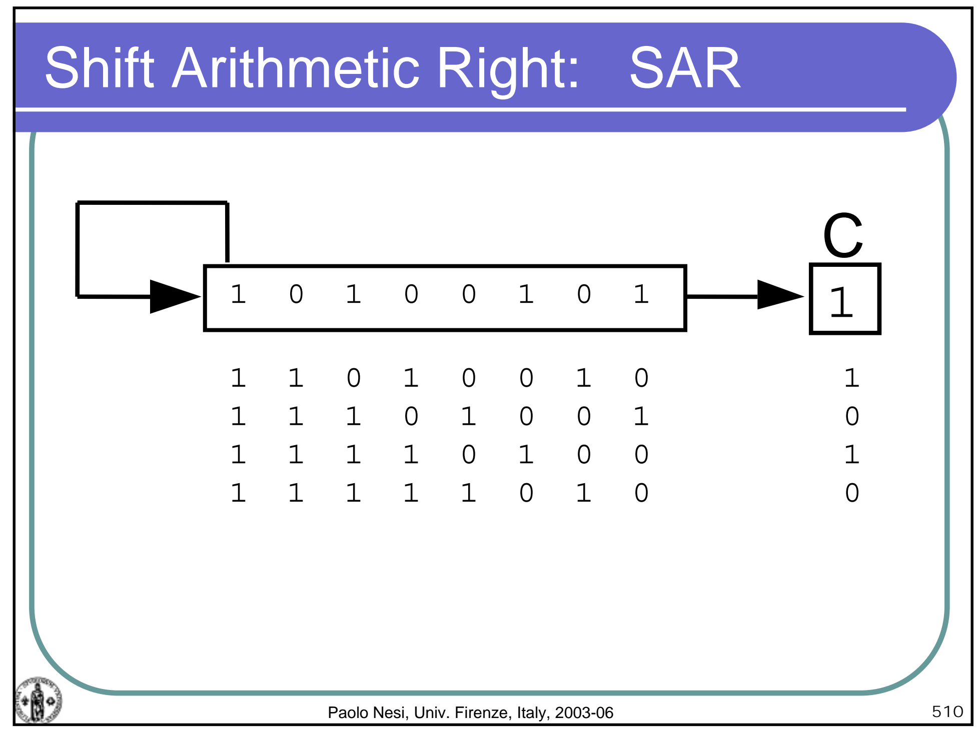
- Supponiamo (CL) = 3

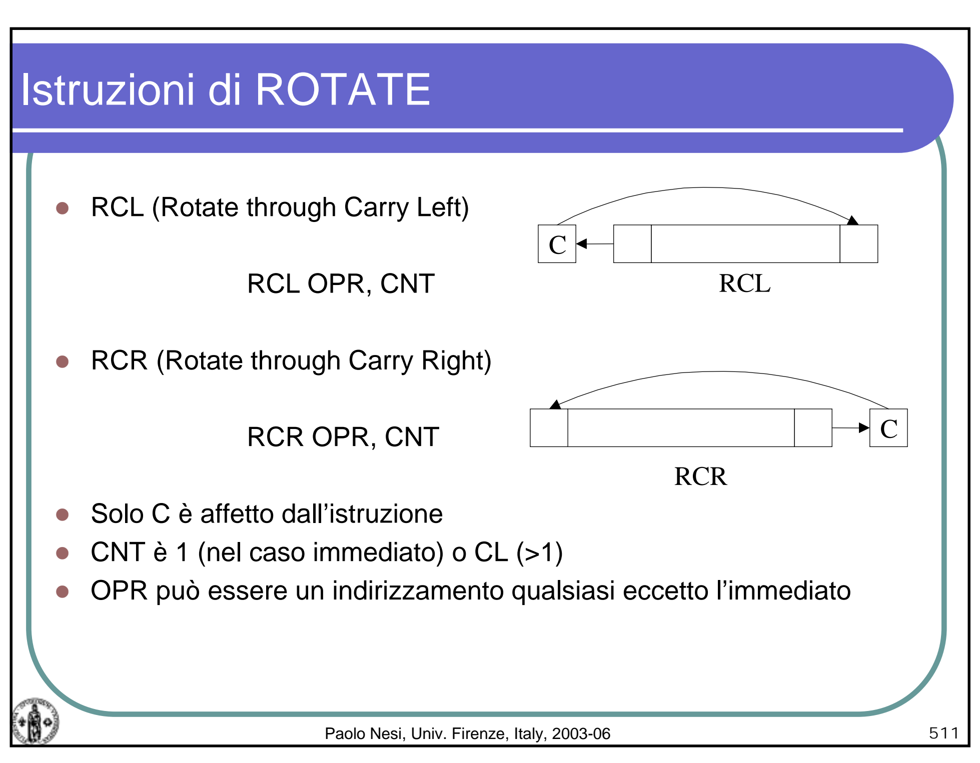
MOV CL, 3
SHL AH, CL oppure SAL AH, CL => (AH) = 10011000 e C=0

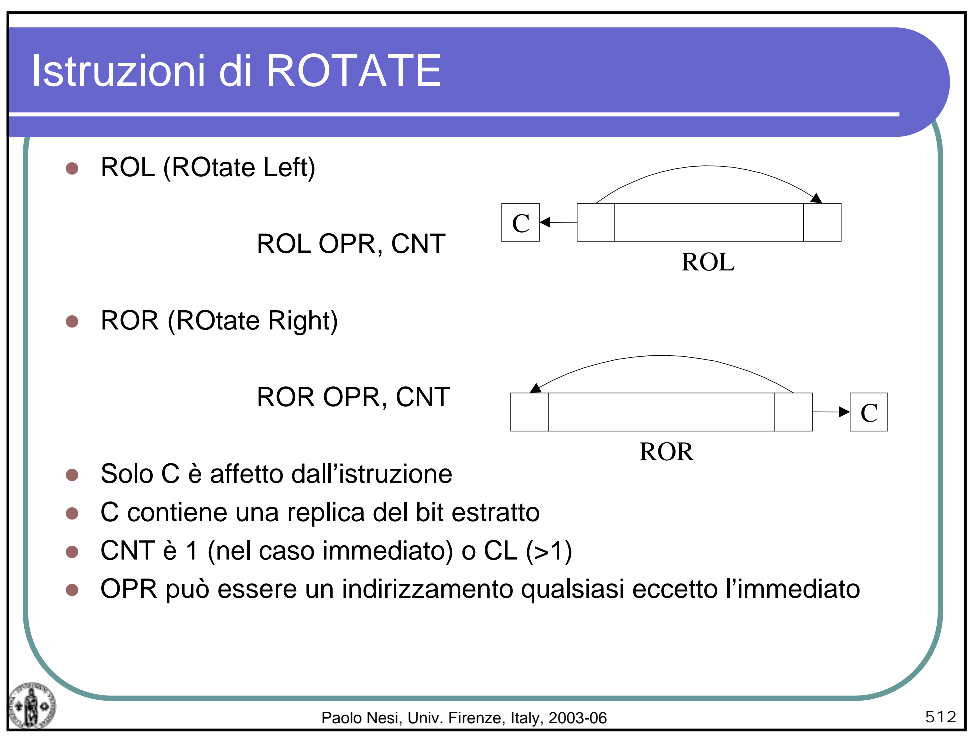
- Supponiamo (AH) = 10010011
  SHR AX, 1 ==> (AH) = 01001001 C=1

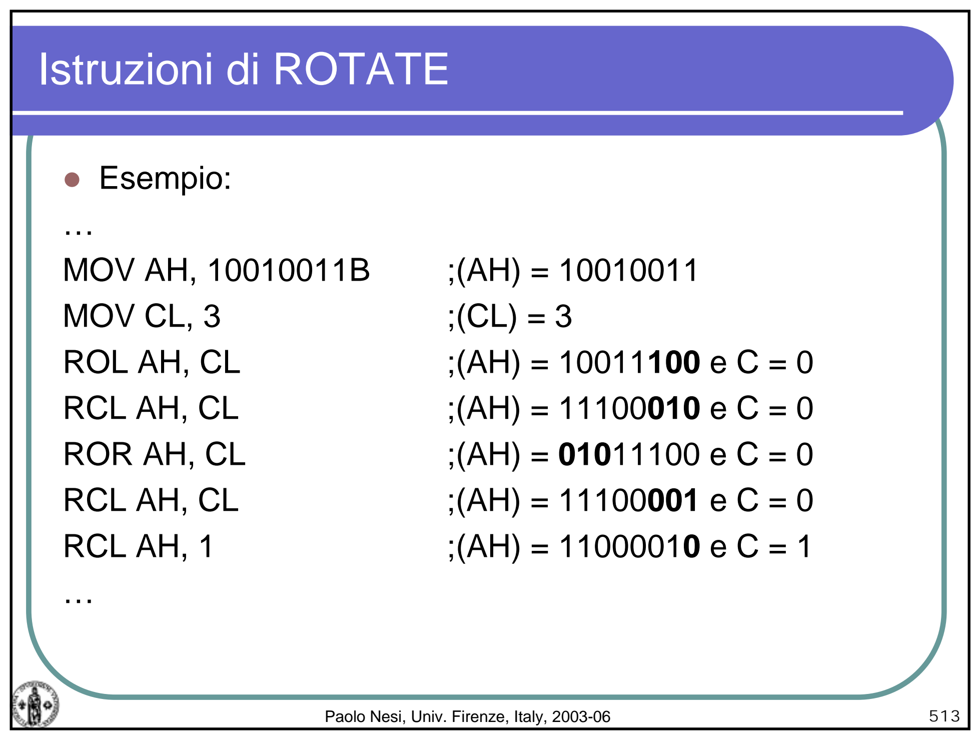

Istruzioni di SHIFT
Esempio:
Supponiamo (AH) = 10010011
SAR AH, 1 => (AH) = 11001001 e C=1
Supponiamo (CL) = 3
SHR AH, CL => (AH) = 00010010 e C=0
SAR AX, CL ==> (AH) = 11110010 C=0
Paolo Nesi, Univ. Firenze, Italy, 2003-06
508


Shift Left: SHL AL,1
Accumulator AL
carry
Binary
Hex
0 1 0 1 0 1 0 1 0 AA
1 0 1 0 1 0 1 0 0 54
0 1 0 1 0 1 0 0 0 A8
1 0 1 0 1 0 0 0 0 50
0 1 0 1 0 0 0 0 0 A0
1 0 1 0 0 0 0 0 0 40
0 1 0 0 0 0 0 0 0 80
1 0 0 0 0 0 0 0 0 00
0 0 0 0 0 0 0 0 0 00
Paolo Nesi, Univ. Firenze, Italy, 2003-06
509


Shift Arithmetic Right: SAR
C
1 0 1 0 0 1 0 1
1
1 1 0 1 0 0 1 0
1
1 1 1 0 1 0 0 1
0
1 1 1 1 0 1 0 0
1
1 1 1 1 1 0 1 0
0
Paolo Nesi, Univ. Firenze, Italy, 2003-06
510


Istruzioni di ROTATE
RCL (Rotate through Carry Left)
RCL OPR, CNT
C
RCL
RCR (Rotate through Carry Right)
RCR OPR, CNT
C
RCR
Solo C è affetto dall'istruzione
CNT è 1 (nel caso immediato) o CL (>1)
OPR può essere un indirizzamento qualsiasi eccetto l'immediato
Paolo Nesi, Univ. Firenze, Italy, 2003-06
511


Istruzioni di ROTATE
ROL (ROtate Left)
ROL OPR, CNT
C
ROL
ROR (ROtate Right)
ROR OPR, CNT
C
ROR
Solo C è affetto dall'istruzione
C contiene una replica del bit estratto
CNT è 1 (nel caso immediato) o CL (>1)
OPR può essere un indirizzamento qualsiasi eccetto l'immediato
Paolo Nesi, Univ. Firenze, Italy, 2003-06
512


Istruzioni di ROTATE
Esempio:
…
MOV AH, 10010011B ;(AH) = 10010011
MOV CL, 3 ;(CL) = 3
ROL AH, CL ;(AH) = 10011100 e C = 0
RCL AH, CL ;(AH) = 11100010 e C = 0
ROR AH, CL ;(AH) = 01011100 e C = 0
RCL AH, CL ;(AH) = 11100001 e C = 0
RCL AH, 1 ;(AH) = 11000010 e C = 1
…
Paolo Nesi, Univ. Firenze, Italy, 2003-06
513

Definizione di Costanti

- Formato:

  <nome costante> EQU <espressione>

  - Definiscono costanti simboliche durante l'assemblaggio.
  - Non possono essere modificate nel corso del programma
  - <espressione> può essere un'espressione intera, una stringa di 1 o 2 caratteri o un indirizzo

- Esempio

```
COLUMN      EQU 80
ROW         EQU 25
SCREEN      EQU COLUMN*ROW
MOV AX, ROW ;ROW vale come dato => Ind. Immediato
            ;la conversione a 8 o a 16 bit la fa
            ;l'assemblatore
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 514

Definizione di Costanti

- Esempi:

  K EQU 1024

Ogni volta che compare K nel codice questa è considerata come fosse scritto "1024".

```
VETT1 DB 1024 DUP(?)
VETT2 DB 2* 1024 DUP(?)
VETT3 DB 3* 1024 DUP(?)
```

Se si definisce il simbolo K per mezzo di EUQ:

```
VETT1 DB K DUP(?)
VETT2 DB 2* K DUP(?)
VETT3 DB 3* K DUP(?)
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 515

La direttiva LABEL

- Permette di associare un'etichetta ad una locazione di memoria contente o un dato o un'istruzione
- è utilizzata nei salti
- Sintassi:

  nome LABEL tipo

  dove tipo può essere:
  - per le etichette vere e proprie NEAR, FAR (salti)
  - per le variabile BYTE, WORD o DWORD

Paolo Nesi, Univ. Firenze, Italy, 2003-06 516

La direttiva LABEL

- Label usata come variabile:

  BETA LABEL BYTE

  ALFA DW ?

BETA punta alla parte bassa di ALFA

Se vogliamo caricare AX a 16 bit:

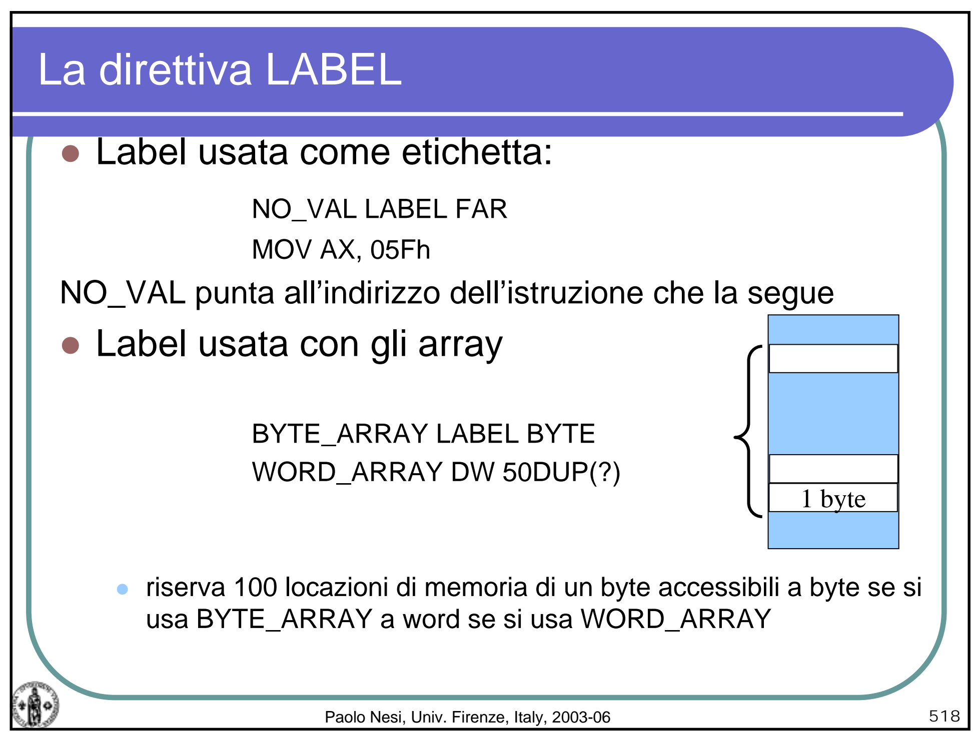
MOV AX, ALFA

Se vogliamo la parte bassa di ALFA:

MOV AL, BETA

Con ALFA si accede per dati di tipo WORD e con BETA di tipo BYTE

Paolo Nesi, Univ. Firenze, Italy, 2003-06 517



Operatore OFFSET

- Formato:

  OFFSET <variabile o label>

  - L'operatore OFFSET restituisce il valore dell'offset di una variabile o della label
  - Può essere usato in alternativa all'istruzione LEA
  - Esempio:

```
MOV AX, OFFSET VAR
LEA AX, VAR
```

Var è una variabile => Sono equivalenti

Paolo Nesi, Univ. Firenze, Italy, 2003-06 519

Operatore OFFSET

- L'operatore OFFSET può essere usato con operandi indirizzati indirettamente:

```
MOV AX, OFFSET VAR[SI] ;ERRORE!!
```

Si può ovviare in questo modo:

```
MOV AX, OFFSET VAR
ADD AX, SI
```

oppure:

```
LEA AX, VAR[SI]
```


Operatori TYPE, LENGTH e SIZE

- Formato:

  TYPE <variabile>
  LENGHT <variabile>
  SIZE <variabile>

  - TYPE restituisce il numero di byte dell'operando, applicato ad una label: -1 se NEAR, -2 se FAR
  - LENGHT restituisce il numero di unità allocate per l'operando
  - SIZE restituisce lo spazio utilizzato dall'operando vale cioè:

    SIZE = LENGHT * TYPE

Operatori TYPE, LENGTH e SIZE

- LENGHT funziona correttamente solo con variabili allocate con DUP. Negli altri casi restituisce il valore 1.
- Esempio:

```
EXP     DW    5 DUP(?)
EXP2    DW    1, 2, 3, 4, 5
        MOV AX, LEGHT EXP   ;(AX) = 05H
        MOV BX, TYPE EXP    ;(BX) = 02H
        MOV CX, SIZE EXP         ;(CX) = 0AH
        MOV DX, LENGHT EXP2      ;(DX) = 01H
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 522

Operatore SEG

- Formato:

SEG <variabile>

Funzionamento:

- L’operatore restituisce l’indirizzo di inizio del segmento cui appartiene l’operando <variabile>

Esempio:

```
LEA SI, STR
MOV AX, SEG STR
MOV DS, AX
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 523

Operatore PTR

- Formato:

  tipo PTR <variabile>±<espressione costante>

Funzionamento:

- Tipo può valere: BYTE, WORD o DWORD
- L'operatore forza l'assemblatore a modificare per l'istruzione corrente il tipo del dato rappresentato da <variabile>±<espressione costante

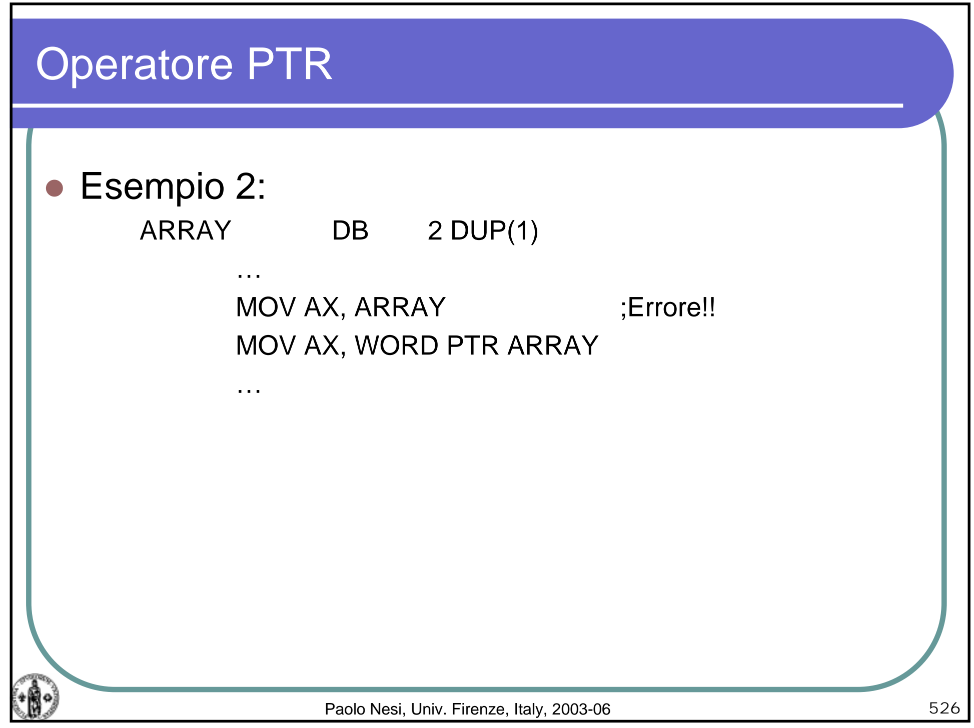
Operatore PTR

- Esempio 1:

```
    …
    MOV OP1+1,AX     } Discordanza sui tipi e
    MOV OP2,AL       } pertanto non assemblabile
    …
OP1 DB 1,2
OP2 DW 1010,2020
```

- per essere assemblato, può essere modificato in:

```
…                                  ;Se (AX) = 0005H
MOV WORD PTR OP1+1, AX             ;(OP+1) = 05H e (OP2) = 00H
MOV BYTE PTR OP2, AL     ;(OP2) = 05H
…
```



Istruzioni di Salto

- Durante l'esecuzione le istruzioni vengono eseguite in ordine sequenziale così come appaiono nel programma
- Utilizzando le istruzioni di salto (*jump*) è possibile forzare il processore a continuare l'esecuzione senza eseguire l'istruzione successiva, ma saltando ad una istruzione diversa
- In Assembly ci sono 2 tipi di salto:
  - Salto Condizionato
  - Salto Incondizionato

Paolo Nesi, Univ. Firenze, Italy, 2003-06 527

Istruzioni di Salto

- Effettuare un salto vuol dire cambiare il contenuto del registro IP in modo che esso punti all'istruzione cui si vuole saltare.
- Con il modello di memoria segmentato dell'8086 l'indirizzo di un'istruzione si ottiene a partire da CS e da IP
- Se il salto avviene all'interno del segmento puntato da CS (SALTO INTRASEGMENTO) viene modificato solo IP
- Se il salto avviene all'interno di un altro segmento (SALTO INTERSEGMENTO) viene cambiato anche CS

Paolo Nesi, Univ. Firenze, Italy, 2003-06 528

Istruzioni di Salto

- In generale i salti possono coinvolgere meccanismi di indirizzamento diretto e indiretto
- Diretto:
  - si fa uso di una LABEL che riferendosi ad un indirizzo definisce l'istruzione cui è associata
  - la label consente di definire così uno scostamento
- Indiretto:
  - si fa uso di indirizzamenti indiretti (registro o memoria) per ottenere l'indirizzo dove saltare

Paolo Nesi, Univ. Firenze, Italy, 2003-06 529

Modi di Indirizzamento per i salti

- INTRASEGMENTO
  - **Intrasegmento diretto:** l'EA (Effective Address) cui saltare si ottiene come somma di (IP) e di uno scostamento a 8 bit (**salto corto**) o a 16 bit (**salto vicino**). Può essere usato sia per il salto CONDIZIONATO che per quello INCONDIZIONATO. Per il primo salto però vale solo quello corto.
  - **Intrasegmento indiretto:** si usa solo nel salto incondizionato. L'EA cui saltare si ottiene sostituendo ad (IP) il contenuto di un registro o di una locazione di memoria indirizzata con uno dei modi di indirizzamento per i dati eccetto quello l'immediato.
- INTERSEGMENTO (solo per salti incondizionati)
  - **Intersegmento diretto:** vengono sostituiti (IP) e (CS) con un offset ed un segmento forniti direttamente dall'istruzione di salto
  - **Intersegmento indiretto:** si sostituisce (IP) e (CS) con il contenuto di due celle di memoria consecutive, indirizzate con uno dei modi per i dati ad esclusione del tipo registro e immediato.

Salti Condizionati

- Definizione di uno scostamento a 8 bit con segno (in complemento a 2 con range [-128, 127]) che deve essere sommato ad (IP) per ottenere l'EA dell'istruzione cui saltare
- Lo scostamento è determinato usando il meccanismo delle Label
- Scostamento >0 salto in avanti, indietro <0

Salti Condizionati

- Formato:

    mnemonico   label±espressione costante

- Lo mnemonico è l'istruzione vera e propria
- label±espressione costante deve rappresentare un offset tra -128 e 127
- L'istruzioni CMP è utile per il test di condizioni, in quanto modifica il registro dei flags
- I salti non modificano i flags

Paolo Nesi, Univ. Firenze, Italy, 2003-06 532

Salti Condizionati

Ipotesi: Ogni istruzione assembly viene codificata in 2 byte

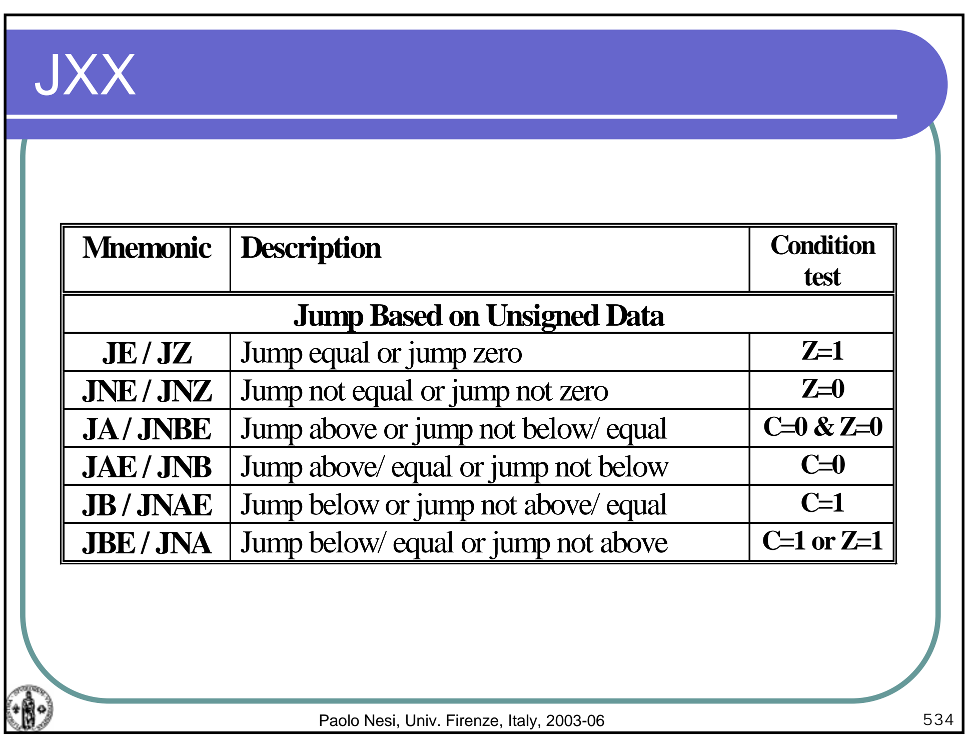
- Esempio:

Effective Address

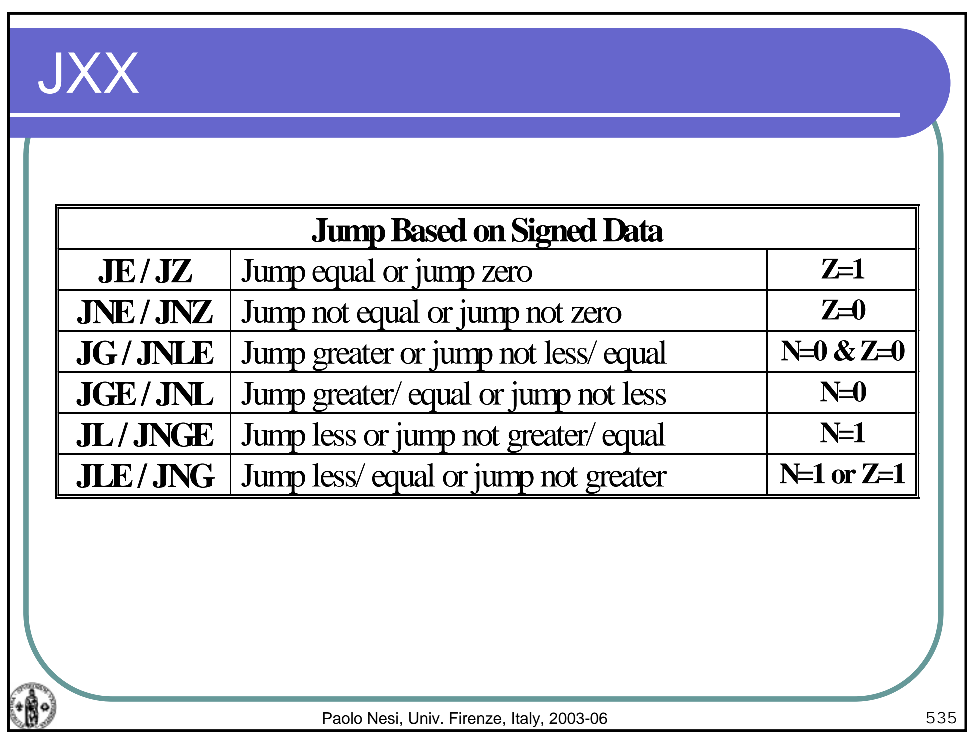
```
0050        START1:   INC CX
0052                  ADD AX, [BX]
0054                  JNS    START1
0056                  MOV RES, CX
```

- Al passo EA = 54 viene caricata l'istruzione JNS START1 ed (IP) viene incrementato di 2 al valore 0056
- Se il flag S è 0, JNS (Jump if Not Sign) aggiunge ad (IP) uno scostamento di -6 per poter raggiungere l'istruzione INC CX
- A livello di codice macchina, sarà l'assemblatore che tradurrà l'istruzione di salto in un codice a 2 byte di cui uno riservato per lo scostamento -6 ovvero FAH.

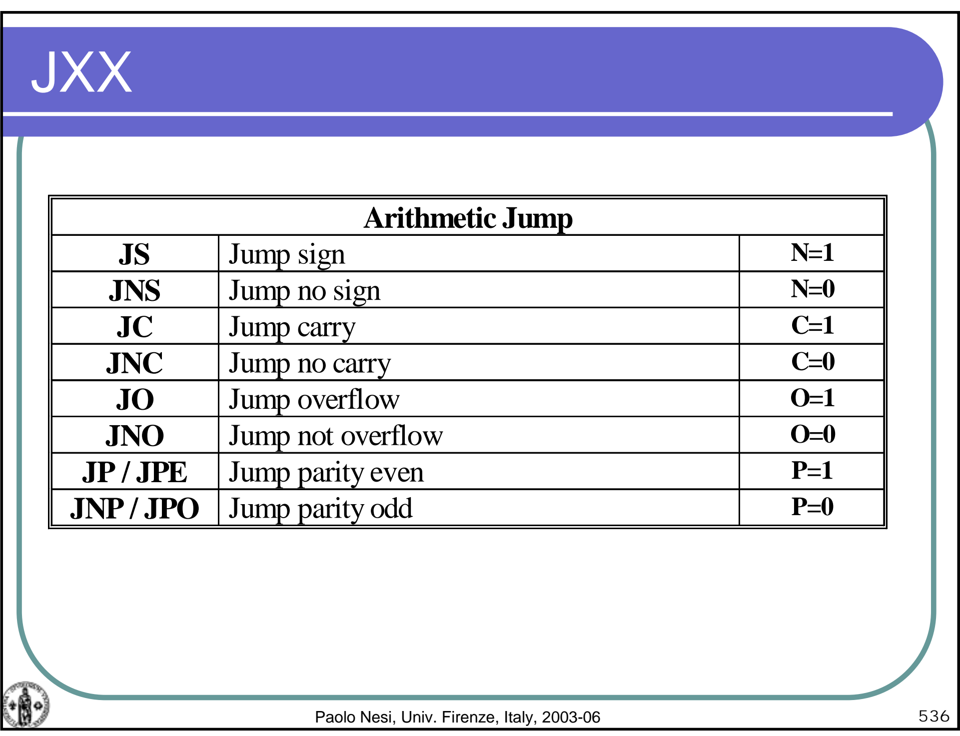
Paolo Nesi, Univ. Firenze, Italy, 2003-06 533

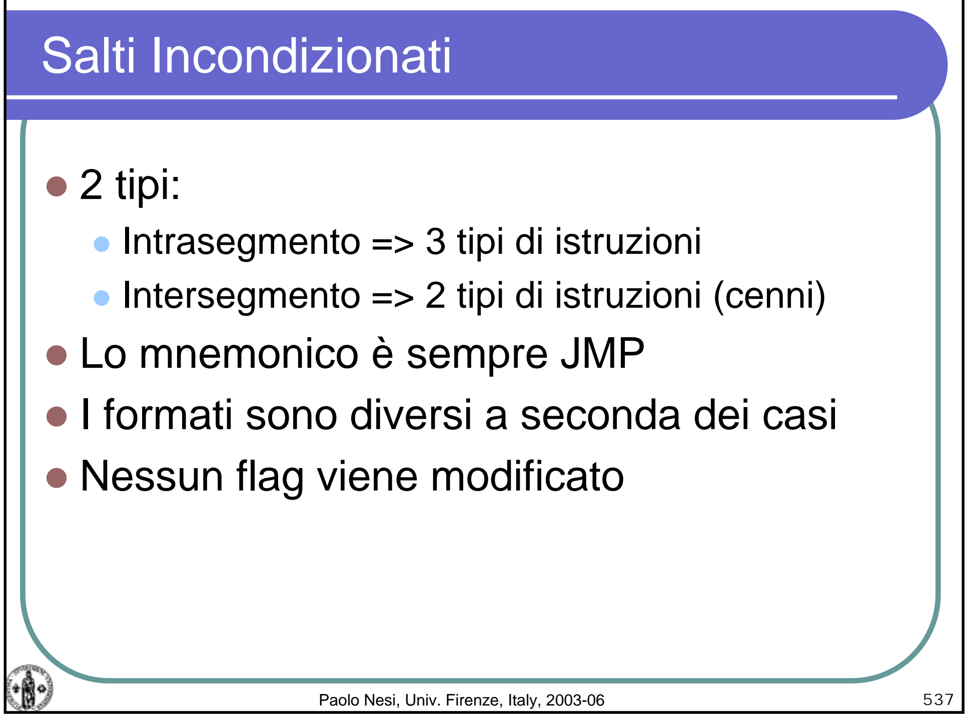


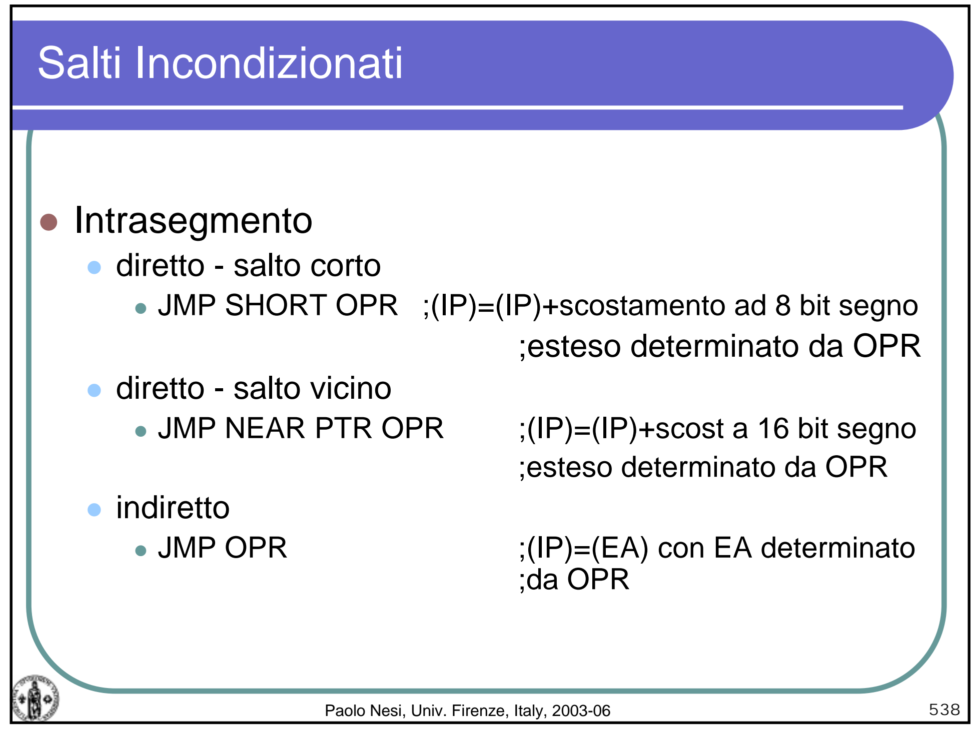
Mnemonic	Description	Condition test
Jump Based on Unsigned Data		
JE / JZ	Jump equal or jump zero	Z=1
JNE / JNZ	Jump not equal or jump not zero	Z=0
JA / JNBE	Jump above or jump not below/ equal	C=0 & Z=0
JAE / JNB	Jump above/ equal or jump not below	C=0
JB / JNAE	Jump below or jump not above/ equal	C=1
JBE / JNA	Jump below/ equal or jump not above	C=1 or Z=1

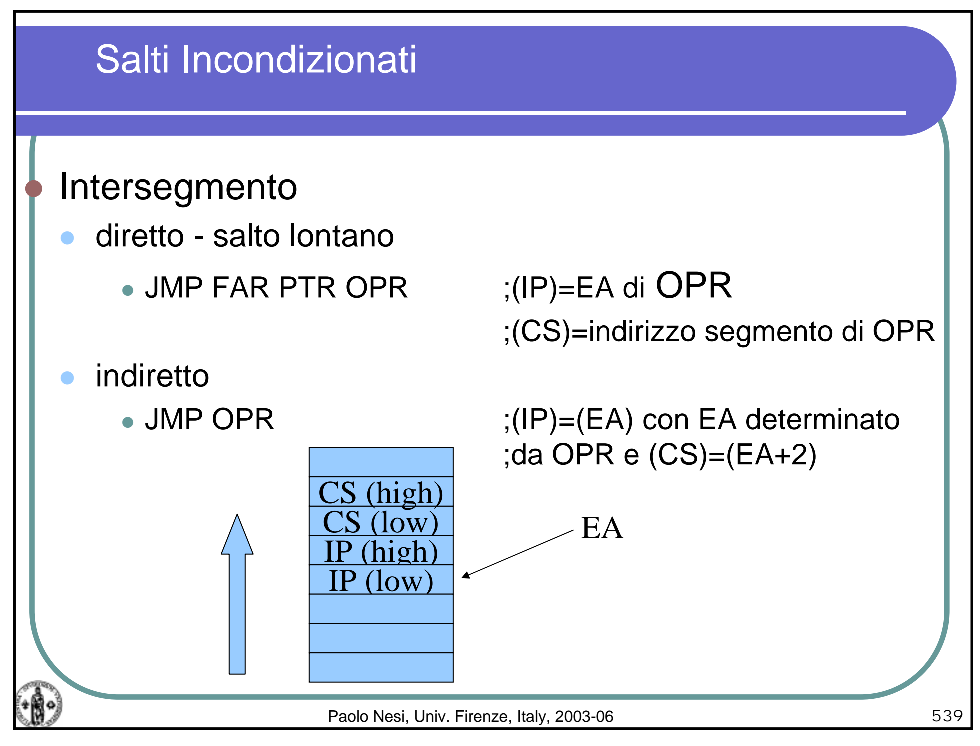


Jump Based on Signed Data		
JE / JZ	Jump equal or jump zero	Z=1
JNE / JNZ	Jump not equal or jump not zero	Z=0
JG / JNLE	Jump greater or jump not less/ equal	N=0 & Z=0
JGE / JNL	Jump greater/ equal or jump not less	N=0
JL / JNGE	Jump less or jump not greater/ equal	N=1
JLE / JNG	Jump less/ equal or jump not greater	N=1 or Z=1


JXX
Arithmetic Jump
JS | Jump sign | N=1
JNS | Jump no sign | N=0
JC | Jump carry | C=1
JNC | Jump no carry | C=0
JO | Jump overflow | O=1
JNO | Jump not overflow | O=0
JP / JPE | Jump parity even | P=1
JNP / JPO | Jump parity odd | P=0
Paolo Nesi, Univ. Firenze, Italy, 2003-06
536


Salti Incondizionati
2 tipi:
Intrasegmento => 3 tipi di istruzioni
Intersegmento => 2 tipi di istruzioni (cenni)
Lo mnemonico è sempre JMP
I formati sono diversi a seconda dei casi
Nessun flag viene modificato
Paolo Nesi, Univ. Firenze, Italy, 2003-06
537


Salti Incondizionati
Intrasegmento
diretto - salto corto
JMP SHORT OPR ;(IP)=(IP)+scostamento ad 8 bit segno
;esteso determinato da OPR
diretto - salto vicino
JMP NEAR PTR OPR ;(IP)=(IP)+scost a 16 bit segno
;esteso determinato da OPR
indiretto
JMP OPR ;(IP)=(EA) con EA determinato
;da OPR
Paolo Nesi, Univ. Firenze, Italy, 2003-06
538


Salti Incondizionati
Intersegmento
diretto - salto lontano
JMP FAR PTR OPR ;(IP)=EA di OPR
;(CS)=indirizzo segmento di OPR
indiretto
JMP OPR ;(IP)=(EA) con EA determinato
;da OPR e (CS)=(EA+2)
CS (high)
CS (low)
IP (high)
IP (low)
EA
Paolo Nesi, Univ. Firenze, Italy, 2003-06
539

Implementare un IF-ELSE

char n;
if (n=='7')
do_it();
else
do_that();

```
;if (n=='7')
  cmp n,'7'
  jne else_
;then-part
  call do_it
  jmp short endif
else_:
  call do_that
endif:
```


Implementare un WHILE

int n;
while (n>0)
n-=2;

```
;while (n>0)
while_:
  cmp n,0
  jle end_while
;loop-body
  sub n,2
  jmp while_
end_while:
```

Valutazioni di espressioni logiche - AND

char n; int w,x;
if (n>='A' && w==x)
whatever();

```
;if(n>='A'&& w==x)
   cmp n,'A'
   jl no_go
   mov ax,w
   cmp ax,x
   jne no_go
;then-part
   call whatever
no_go:
```


Valutazioni di espressioni logiche - OR

char n,k;
unsigned int w;
…
if (n<>k || w<=10)
whatever();

```
;if(n<>k||w<=10)
   mov ah,n
   cmp ah,k
   jne then_
   cmp w,10
   ja end_if
then_:
   call whatever
end_if:
```

Cicli

- Un possibile ciclo con test a posteriori usando i salti condizionati:

```
            …
            MOV CX, N       ;uso CX come contatore
INIZIO: …                   ;inizio del corpo del loop
            …
FINE:       …               ;fine del corpo del loop
            DEC CX;decremento di CX
            JNZ INIZIO      ;salto ad inizio se il conteggio non è
                            ;completo
```

Nota: N rappresenta in questo caso l'EA della cella di memoria contenente il numero di volte che il ciclo deve essere ripetuto.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 544

Istruzione LOOP

- Istruzioni per la gestione di cicli che utilizzano implicitamente il registro CX decrementandolo di una unità ad ogni ciclo.
- Il ciclo viene ripetuto fino a che il registro CX non è zero.
- Formato:

LOOP label

- **Esempio:**

```
                MOV CX, 100         ;numero di iterazioni 100
        INIZIO: <istruzione 1>
                <istruzione 2>
                …
                <istruzione n>
                LOOP INIZIO         ;il blocco di istruzioni
                                    ;è eseguito 100 volte
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 545

Istruzione LOOP

Esempio:

for (x=9;x>0;x--)

n+=x;

```
                                        ;for(x=9;x>0;x--)
            MOV CX,9
TOP_LOOP:   ADD N,CX                    ;N=N+X
            LOOP TOP_LOOP
```


Istruzioni LOOP

- L'assembler fornisce altre 5 istruzioni per realizzare loop condizionati:
- LOOPZ (loop while zero)
  - LOOPZ OPR ; si cicla finché (CX)≠0 AND Z=1
- LOOPE (loop while equal)
  - LOOPE OPR ; si cicla finché (CX)≠0 AND Z=1
- LOOPNZ (loop while not-zero)
  - LOOPNZ OPR ; si cicla finché (CX)≠0 AND Z=0
- LOOPNE (loop while not-equal)
  - LOOPNE OPR ; si cicla finché (CX)≠0 AND Z=0
- JCXZ (Jump if CX is Zero)
  - JCXZ OPR ;si cicla finchè (CX)=0

Istruzioni LOOP

- Le istruzioni LOOPE e LOOPNE sono normalmente usate a seguito dell'istruzione CMP e sono logicamente identiche rispettivamente a LOOPZ e LOOPNZ
- Se si verifica la condizione di continuazione del ciclo il contenuto di IP è sostituito dalla somma di (IP) e dello scostamento ad 8 bit (salto corto) rappresentato da OPR (o da LABEL).
- Ad esclusione di JCXZ, tutte le istruzioni di LOOP decrementano (CX)
- Nessun flag viene modificato

Paolo Nesi, Univ. Firenze, Italy, 2003-06 548

Istruzioni LOOP

- **Esempio**: Ricerca in una stringa di L caratteri del carattere spazio, codificato in ASCII con il valore 20h. La stringa è allocata all'indirizzo indicato da ASCII_STR.

```
        …
        MOV CX, L               ;inizializzo il contatore
        MOV SI, -1              ;inizializzo l'indice
        MOV AL, 20h             ;metto in AL il codice da trovare
START:  INC SI                  ;si incrementa l'indice
        CMP ASCII_STR, 20h      ;confronta lo spazio con il carattere puntato
        LOOPNE START ;se non sono uguali cerco ancora e (CX) viene
                     ;decrementato
        JNZ NON_TROVATO         ;se " " non c'è si salta a NON_TROVATO
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 549

Istruzioni LOOP

- Sono possibili LOOP annidati
- In tal caso quando si accede ai loop via via più interni è necessario salvare il contenuto di CX in modo da ripristinare i contatori quando tali loop sono finiti.
- Si può saltare all'interno di un loop

Paolo Nesi, Univ. Firenze, Italy, 2003-06 550

Calcolatori Elettronici

CDL in Ingegneria Elettronica

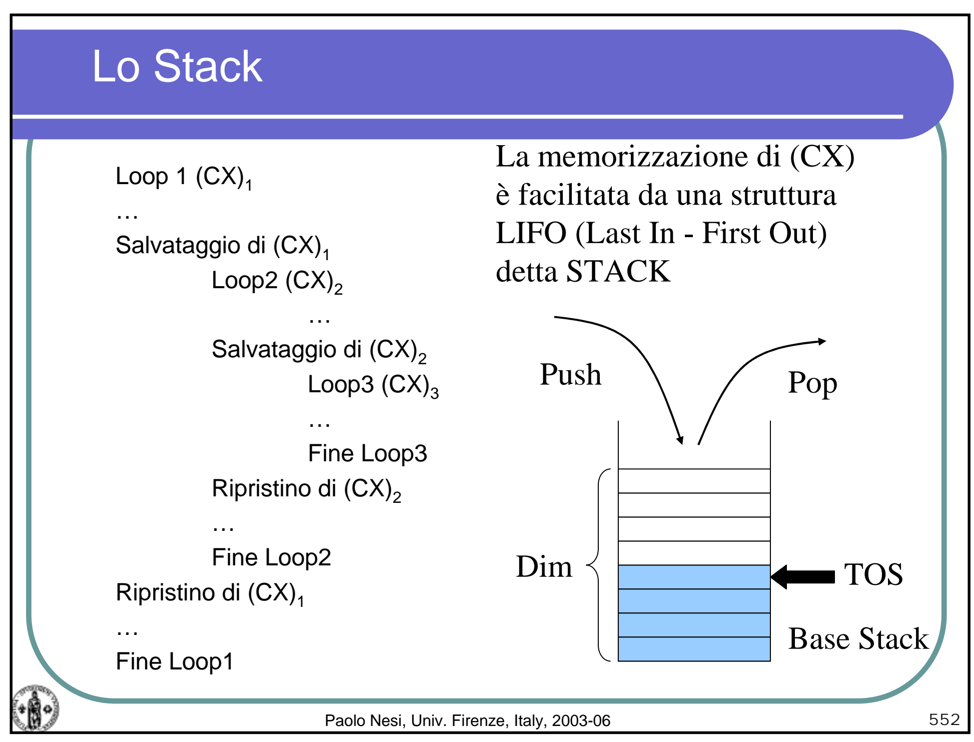
Facolta' di Ingegneria,
Universita' degli Studi di Firenze

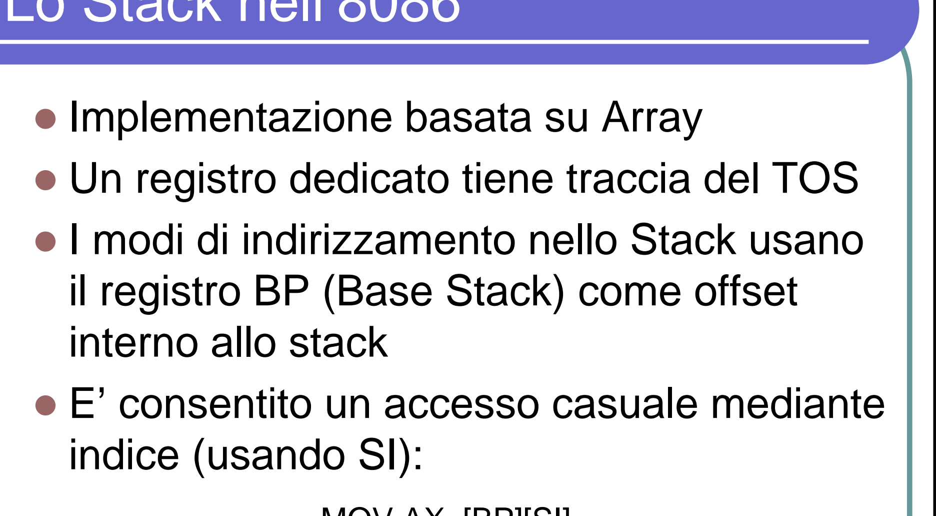
Nuovo Ordinamento

Parte 10: Procedure, Stack ed Interruzioni

Dr. Ing. Ivan Bruno
Prof. Paolo Nesi
http://www.dsi.unifi.it/~ivanb
ivanb@dsi.unifi.it
AA 2005-2006

Paolo Nesi, Univ. Firenze, Italy, 2003-06 551


Lo Stack
Loop 1 (CX)1
…
Salvataggio di (CX)1
Loop2 (CX)2
…
Salvataggio di (CX)2
Loop3 (CX)3
…
Fine Loop3
Ripristino di (CX)2
…
Fine Loop2
Ripristino di (CX)1
…
Fine Loop1
La memorizzazione di (CX)
è facilitata da una struttura
LIFO (Last In - First Out)
detta STACK
Push
Pop
Dim
TOS
Base Stack
Paolo Nesi, Univ. Firenze, Italy, 2003-06
552

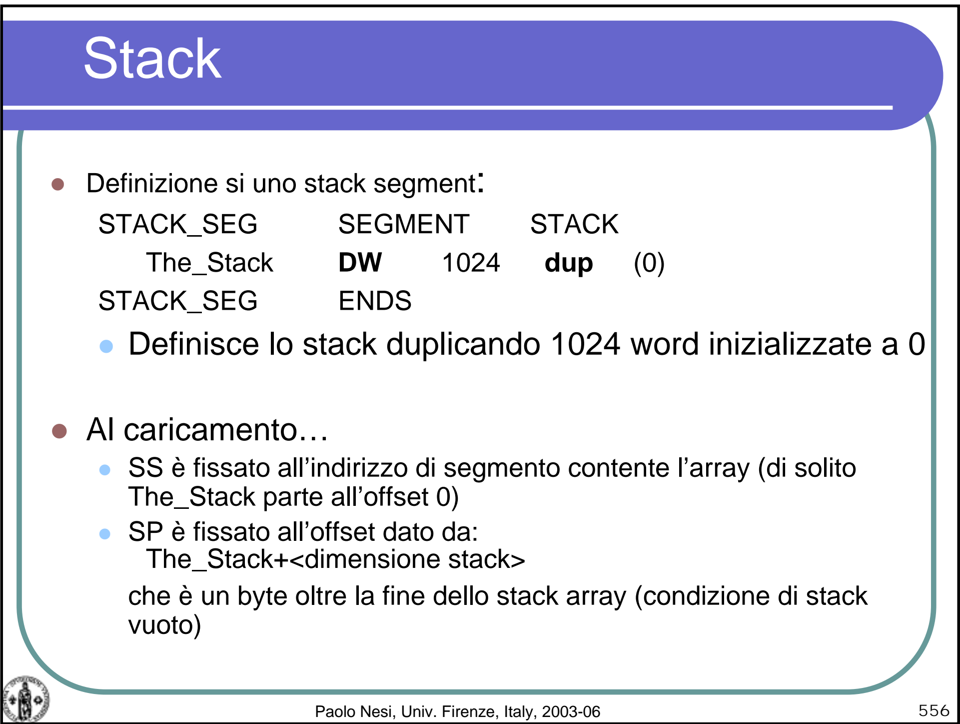

Lo Stack nell'8086
Implementazione basata su Array
Un registro dedicato tiene traccia del TOS
I modi di indirizzamento nello Stack usano il registro BP (Base Stack) come offset interno allo stack
E' consentito un accesso casuale mediante indice (usando SI):
MOV AX, [BP][SI]
Paolo Nesi, Univ. Firenze, Italy, 2003-06
553

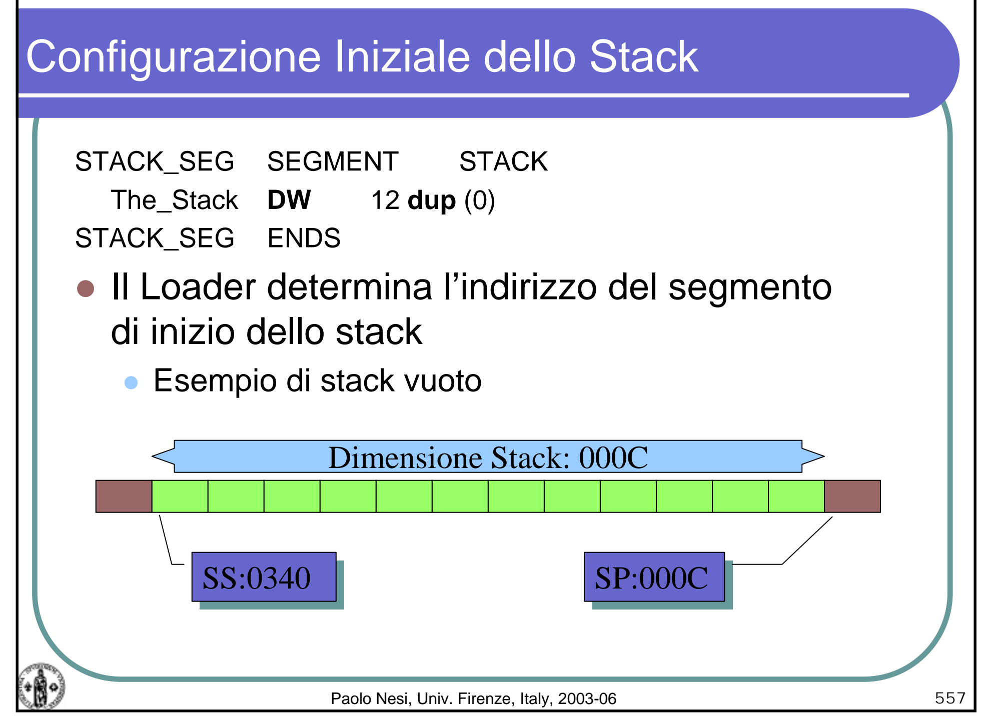
Perché avere uno Stack

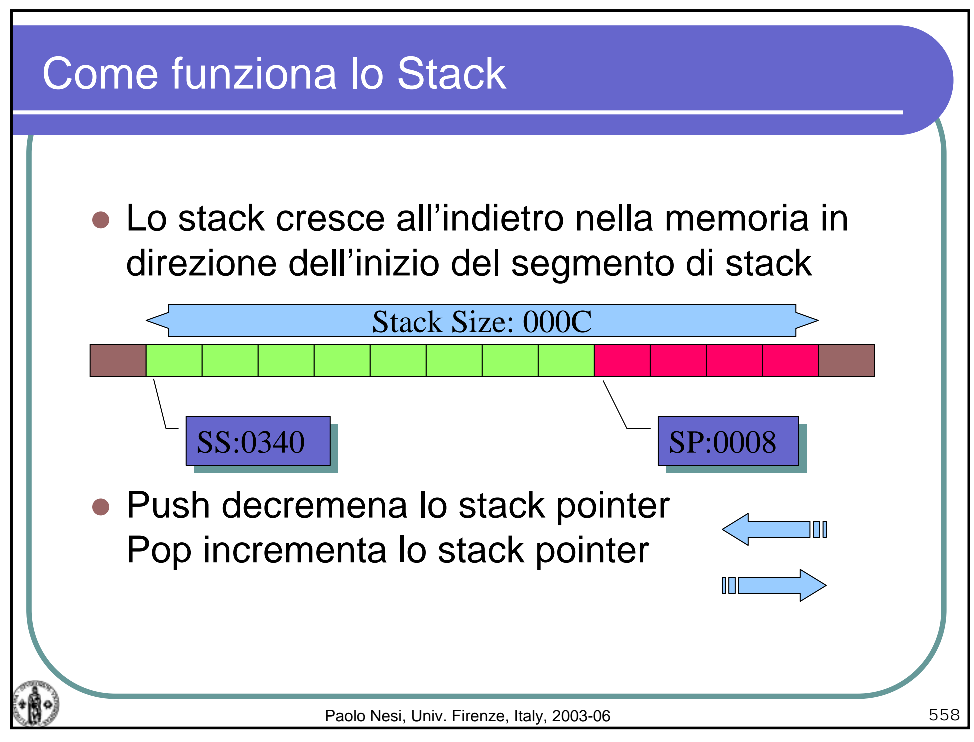
- L'8086 ha istruzioni per gestirlo
- Serve al processore in occasione di interrupts (salvataggio dello stato)
- Le chiamate a Procedure usano lo stack per il successivo ripristino degli indirizzi
- E' conveniente averne uno da usare come memoria temporanea

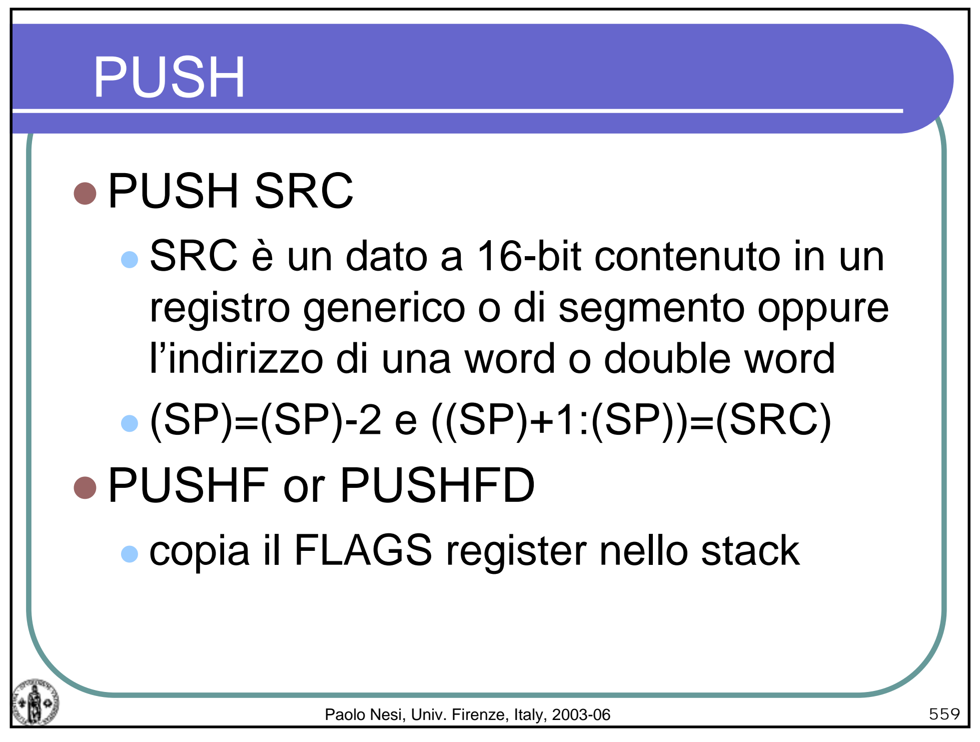
Stack

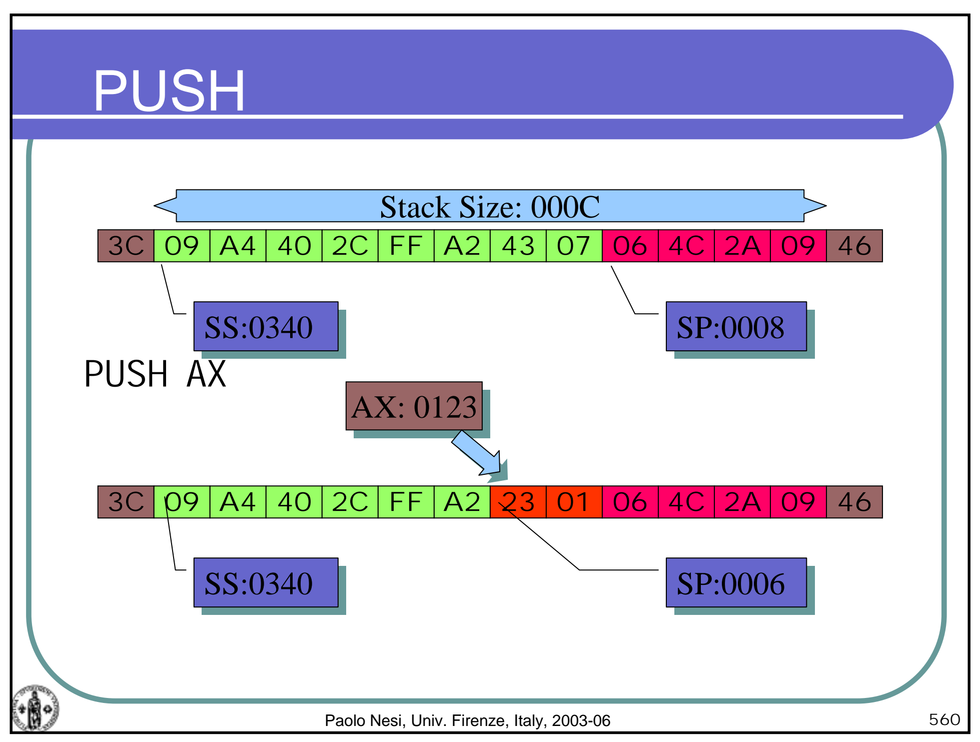
- Tutti i programmi eseguibili devono definire un segmento per lo stack
  - Un array di bytes accessibili attraverso il registro SS e un offset
- SS punta all'inizio dell'area di memoria
- SP è l'offset del TOS (top of the stack)
- Il loader del S.O. inizializza questi registri prima che l'esecuzione del programma abbia inizio

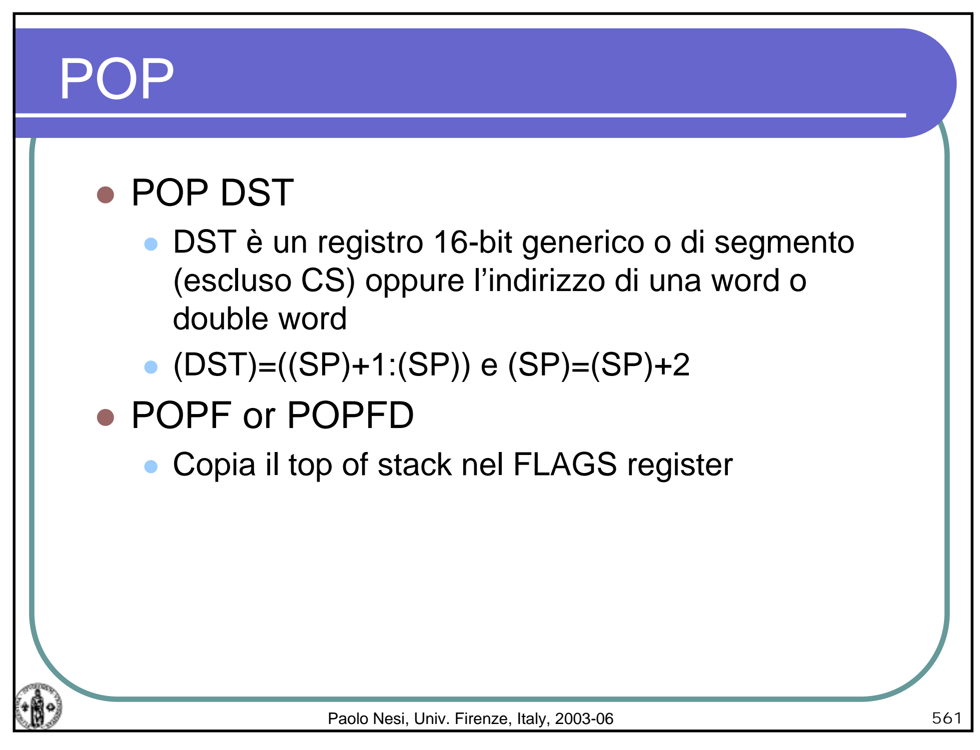

Stack
Definizione si uno stack segment:
STACK_SEG SEGMENT STACK
The_Stack DW 1024 dup (0)
STACK_SEG ENDS
Definisce lo stack duplicando 1024 word inizializzate a 0
Al caricamento…
SS è fissato all'indirizzo di segmento contente l'array (di solito The_Stack parte all'offset 0)
SP è fissato all'offset dato da:
The_Stack+<dimensione stack>
che è un byte oltre la fine dello stack array (condizione di stack vuoto)
Paolo Nesi, Univ. Firenze, Italy, 2003-06
556

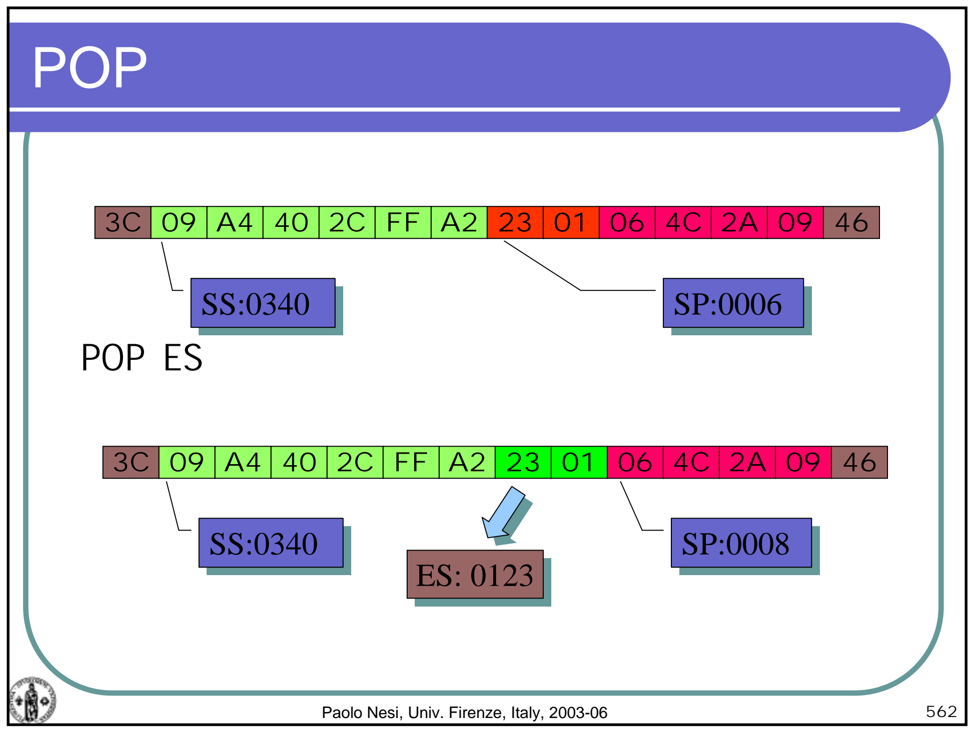

Configurazione Iniziale dello Stack
STACK_SEG SEGMENT STACK
The_Stack DW 12 dup (0)
STACK_SEG ENDS
Il Loader determina l'indirizzo del segmento di inizio dello stack
Esempio di stack vuoto
Dimensione Stack: 000C
SS:0340
SP:000C
Paolo Nesi, Univ. Firenze, Italy, 2003-06
557

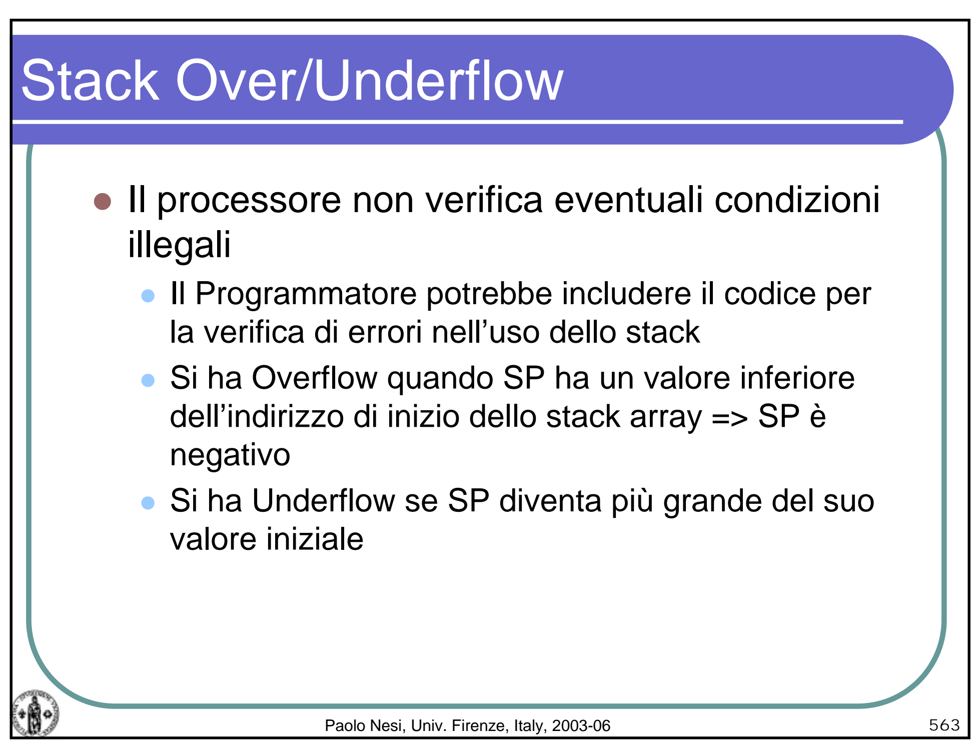

Come funziona lo Stack
Lo stack cresce all'indietro nella memoria in direzione dell'inizio del segmento di stack
Stack Size: 000C
SS:0340
SP:0008
Push decremena lo stack pointer
Pop incrementa lo stack pointer
Paolo Nesi, Univ. Firenze, Italy, 2003-06
558

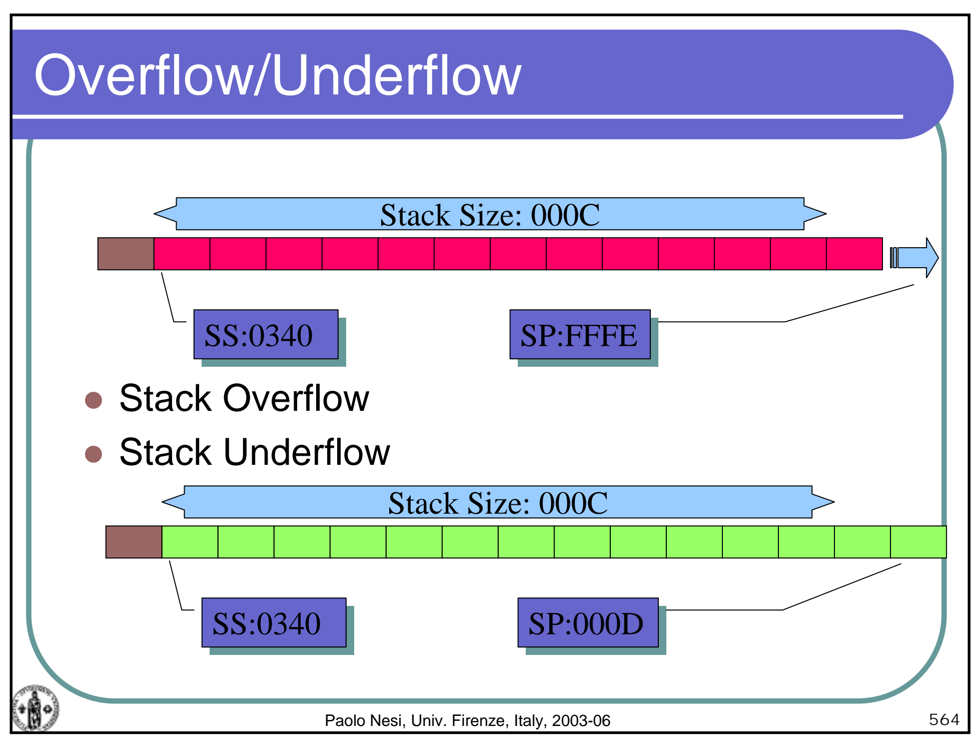

PUSH
PUSH SRC
SRC è un dato a 16-bit contenuto in un registro generico o di segmento oppure l'indirizzo di una word o double word
(SP)=(SP)-2 e ((SP)+1:(SP))=(SRC)
PUSHF or PUSHFD
copia il FLAGS register nello stack
Paolo Nesi, Univ. Firenze, Italy, 2003-06
559


PUSH
Stack Size: 000C
3C 09 A4 40 2C FF A2 43 07 06 4C 2A 09 46
SS:0340
SP:0008
PUSH AX
AX: 0123
3C 09 A4 40 2C FF A2 23 01 06 4C 2A 09 46
SS:0340
SP:0006
Paolo Nesi, Univ. Firenze, Italy, 2003-06
560


POP
POP DST
DST è un registro 16-bit generico o di segmento (escluso CS) oppure l'indirizzo di una word o double word
(DST)=((SP)+1:(SP)) e (SP)=(SP)+2
POPF or POPFD
Copia il top of stack nel FLAGS register
Paolo Nesi, Univ. Firenze, Italy, 2003-06
561


POP
3C 09 A4 40 2C FF A2 23 01 06 4C 2A 09 46
SS:0340
SP:0006
POP ES
3C 09 A4 40 2C FF A2 23 01 06 4C 2A 09 46
SS:0340
ES: 0123
SP:0008
Paolo Nesi, Univ. Firenze, Italy, 2003-06
562


Stack Over/Underflow
Il processore non verifica eventuali condizioni illegali
Il Programmatore potrebbe includere il codice per la verifica di errori nell'uso dello stack
Si ha Overflow quando SP ha un valore inferiore dell'indirizzo di inizio dello stack array => SP è negativo
Si ha Underflow se SP diventa più grande del suo valore iniziale
Paolo Nesi, Univ. Firenze, Italy, 2003-06
563



Procedure

- Una procedura è un insieme di istruzioni cui ci si può riferire all'interno di un segmento di codice
- E' come se le istruzioni fossero presenti effettivamente nel segmento a partire dal punto in cui questa viene invocata
- La chiamata è come un salto all'indirizzo della prima istruzione della procedura e alla fine di questa un altro salto riporta all'istruzione successiva alla chiamata

Paolo Nesi, Univ. Firenze, Italy, 2003-06 565

Procedure

Definizione:

```
proc_nome    PROC     tipo
   ;corpo della procedura
   RET    ;to return to caller
proc_nome    ENDP
```

- *tipo* vale NEAR o FAR
  - il valore di default è NEAR
- Le procedure possono avere uno o più RET

Direttive per la definizione di Procedure

- ***NEAR*** - la procedura può essere chiamata solo all'interno del segmento in cui è stata definita (default)
- ***FAR*** - la procedura può essere chiamata da qualsiasi segmento
- La direttiva PROC definisce l'inizio di una procedura.
- La direttiva ENDP definisce la fine di una procedura.

Chiamata di una Procedura e Ritorno

- Chiamata di una procedura (NEAR)
  CALL *proc_name*
  - push IP nello stack
  - l'indirizzo di *proc_nome* è copiato nell'IP
- Return da una procedura (NEAR)
  RET *[n]*
  - pop top of stack in IP
  - aggiunge *n* a SP (Ritorno con valore optional)

Paolo Nesi, Univ. Firenze, Italy, 2003-06 568

Procedure Far

- Chiamata di una procedura (FAR)
  CALL FAR PTR *proc_name*
  - push CS e IP nello stack
  - l'indirizzo far di *proc_name* è copiato in CS:IP
- Return da una procedure (FAR)
  RET *[n]*
  - pop top of stack in IP e poi pop in CS
  - aggiunge *n* a SP (optional)

Paolo Nesi, Univ. Firenze, Italy, 2003-06 569

Interrupts

- Forniscono un meccanismo mediante il quale è possibile trasferire il controllo del programma ad una routine predisposta a gestire il verificarsi di un particolare evento (interno o esterno al µP).

Esistono due tipi di interrupt

Interrupt Hardware
- Generati dall'esterno (I/O)
- Consentono di gestire eventi asincroni
- Possono essere mascherabili e non mascherabili

Interrupt Software
- Generati durante l'esecuzione del programma
- Direttamente mediante istruzioni esplicite (es INT)
- Indirettamente, nel caso si verifichino particolari eventi all'interno del processore (es divisione per zero,…)

Interrupts

- Le locazioni da 0H a 3FFH contengono l'Interrupt Vector Table con 256 ingressi. Ogni ingresso contiene due valori di 16 bit che forniscono l'indirizzo della routine di servizio dell'interrupt e che vengono caricati nei registri CS e IP quando l'interrupt viene accettato.
- I primi cinque elementi della tabella sono dedicati a particolari tipi di interrupt predefiniti nell'8086. I successivi 27 elementi sono riservati all'hardware del sistema di elaborazione e non devono essere utilizzati. I rimanenti elementi (da 32 a 255) sono disponibili per le routine di servizio e del sistema operativo dell'utente
- Un programma può **anche** generare esplicitamente un interrupt di tipo n, mediante l'istruzione **INT n**

Interrupts

Sull'8086 un interrupt è causato quando il pin **intr** passa dal valore 0 a 1.

- Un interrupt può essere disabilitato o mascherato pulendo l'Interrupt Flag (I) usandol'istruzione **CLI**.
- Gli Interrupts possono essere abilitati con l'istruzione **STI**
- Gli Interrupts sono di solito generati da dispositivi di I/O.
- Quando avviene un interrupt la CPU completa l'istruzione corrente quindi:

(1) Disabilita the maskable interrupt con **CLI** (Previene che l'interrupt routine sia a sua volta interrotta.).

(2) Salva IP, CS e il Flags register nello stack (PUSH)

(3) salta all'indirizzo trovato in memoria alla 4*N, dove N è il numero di interrupt.

(4) Esegue la (ISR) routine di servizio associata all'interrupt

(5) Al termine della routine esegue un istruzione **IRET** la quale ripristina IP, CS e il flags register presi dallo stack (POP) ripristindando così lo stato che la CPU aveva prima dell'arrivo dell'interruzione

Paolo Nesi, Univ. Firenze, Italy, 2003-06 572

Esempi di Interruzioni

- **Interrupt 0** (Divide Error) - segnala un errore durante un operazione di divisione (ad es., divisione per zero)
- **Interrupt 1** (Single Step) - un'istruzione dopo il settaggio di TF-trap flag (permette di eseguire una singola istruzione all'interno di un programma - utilizzato dal debugger)
- **Interrupt 2** (Non-Maskable Interrupt) - è l'interrupt hardware di priorità più alta e non è mascherabile - di norma, è riservato ad eventi importanti e urgenti (ad es., una caduta di tensione, un errore nella memoria, un errore sul bus di sistema)
- **Interrupt 3** (One Byte Interrupt) - utilizzato dal debugger per i breakpoint
- **Interrupt 4** (Interrupt on Overflow) - condizione di overflow

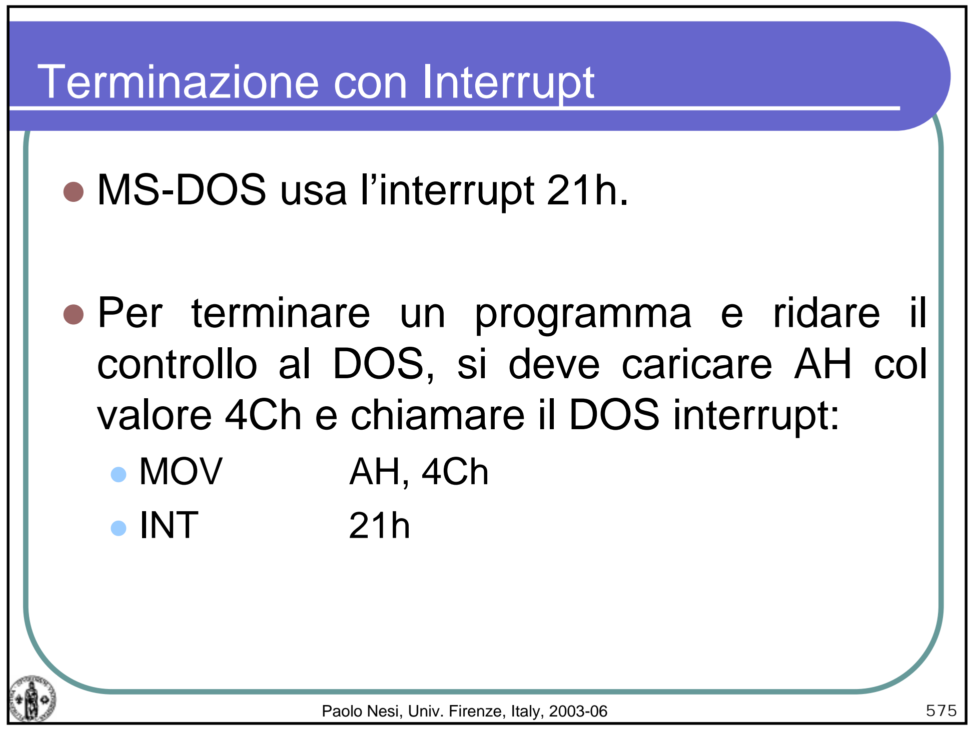
  (OF = 1) e viene eseguita l'istruzione INTO; permette di gestire l'eventuale condizione di overflow

Paolo Nesi, Univ. Firenze, Italy, 2003-06 573

Istruzione INT

- INT instruction
  - **INT** <interrupt number>

Paolo Nesi, Univ. Firenze, Italy, 2003-06 574



Simple Interrupt

- Int 21h fornisce alcuni servizi per la gestione della tastiera e dello schermo

Service No.	Explanation
01h	Keyboard input with echo
02h	Display output
07h	Keyboard input without echo (no check for Ctrl-C)
08h	Keyboard input without echo (check for Ctrl-C)
09h	Display string
4Ch	Terminate program

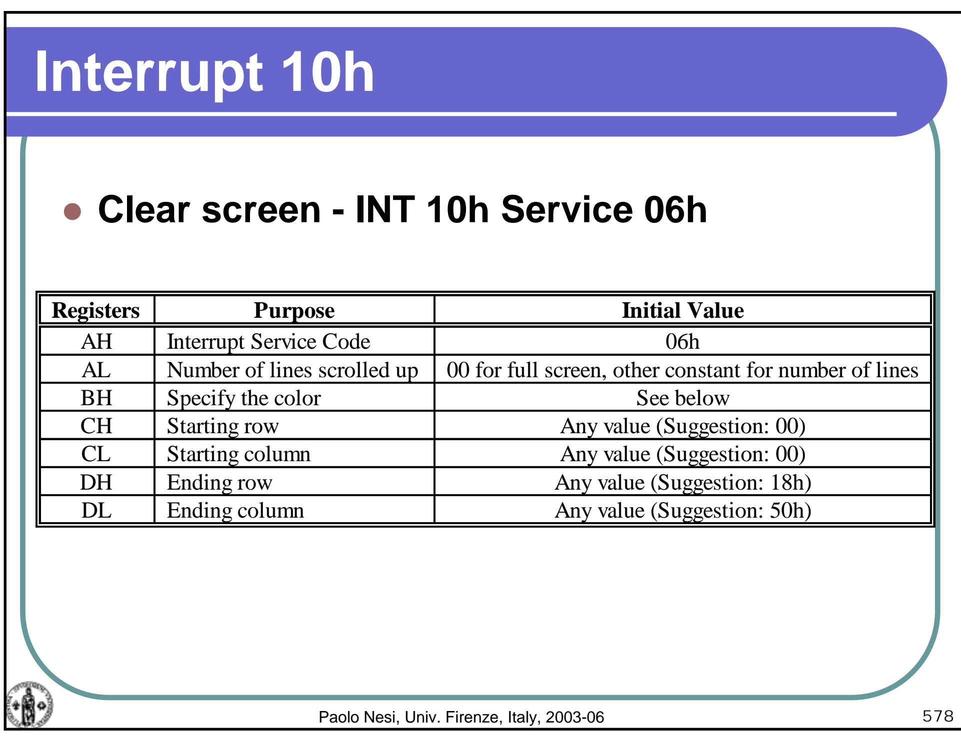
INT *21H*

- Si sceglie la funzione desiderata ponendo un opportunio valore nel registro AH. In base alla funzione scelta si dovranno utilizzare altri registri per passare dei parametri.
- **Funzione di acquisizione di un carattere da tastiera (01H)**

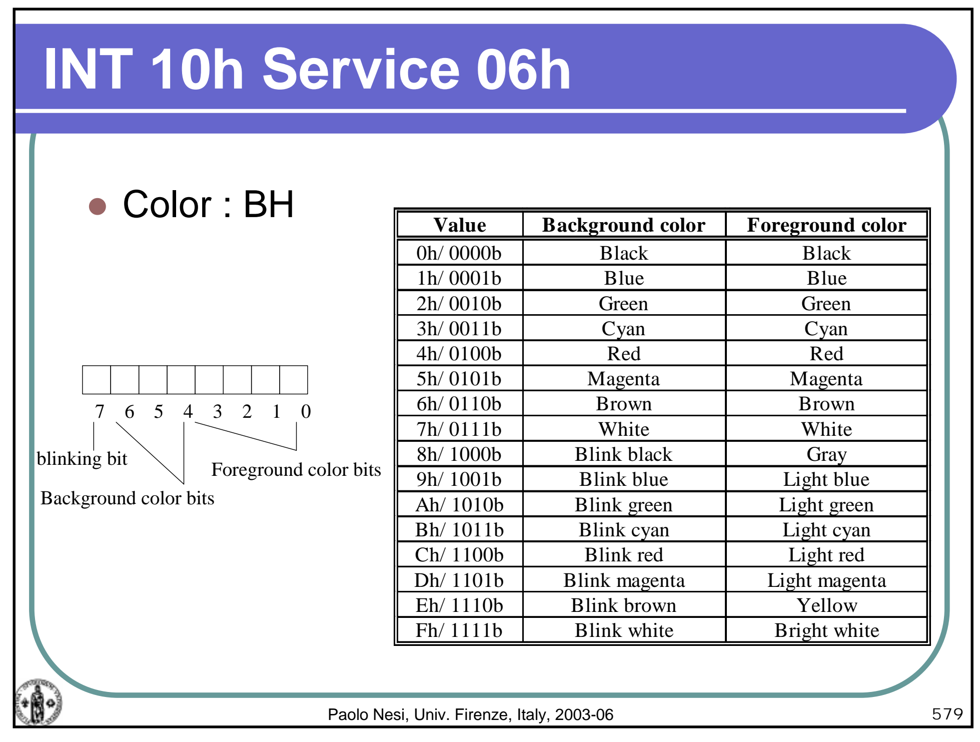
```
MOV AH,01h        ;Seleziona la funzione 01h, di acquisizione da
                  ;tastiera
INT 21h           ;Chiama l'interrupt 21h. Il codice ASCII del
                  ;carattere premuto viene posto in AL
```

- **Funzione di emissione di un carattere su video (02H)**

```
MOV AH, 02h       ;Seleziona la funzione 02H, di
                  ;emissione su video
MOV DL, xxx       ;Mette in DL il codice ASCII da stampare
INT 21h           ;Stampa tale carattere sullo schermo
```



Registers	Purpose	Initial Value
AH	Interrupt Service Code	06h
AL	Number of lines scrolled up	00 for full screen, other constant for number of lines
BH	Specify the color	See below
CH	Starting row	Any value (Suggestion: 00)
CL	Starting column	Any value (Suggestion: 00)
DH	Ending row	Any value (Suggestion: 18h)
DL	Ending column	Any value (Suggestion: 50h)



Value	Background color	Foreground color
0h/ 0000b	Black	Black
1h/ 0001b	Blue	Blue
2h/ 0010b	Green	Green
3h/ 0011b	Cyan	Cyan
4h/ 0100b	Red	Red
5h/ 0101b	Magenta	Magenta
6h/ 0110b	Brown	Brown
7h/ 0111b	White	White
8h/ 1000b	Blink black	Gray
9h/ 1001b	Blink blue	Light blue
Ah/ 1010b	Blink green	Light green
Bh/ 1011b	Blink cyan	Light cyan
Ch/ 1100b	Blink red	Light red
Dh/ 1101b	Blink magenta	Light magenta
Eh/ 1110b	Blink brown	Yellow
Fh/ 1111b	Blink white	Bright white

Setting the cursor INT 10h

- La posizione del cursore determina dove visualizzare il prossimo carattere
- **INT 10h Service 02h** definisce l'operazione di set del cursore
- **BH** definisce il numero di pagina
- **DH** & **DL** definiscono la riga (y) e la colonna (x) per la posizione

Get cursor position - INT 10h

- **INT 10h Service 03h** e fissare il numero di pagina **BH** a 00.
  - MOV AH, 03h
  - MOV BH,00h
  - INT 10h
- dopo l'esecuzione, **DH** conterrà il numero di riga e **DL** il numero di colonna della posizione del cursore.

Read character from the cursor position

- Interrupt **10h** con servizio numero **08h**
  - MOV AH, 08h
  - MOV BH, 00h
  - INT 10h
- Il carattere potrà essere letto nel registro **AL** in rappresentazione ASCII.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 582

Comunicazione con dispositivi I/O

Esegue i trasferimenti dati dallo spazio di I/O verso il processore.

IN registro, ind_porta_IO

dove il registro deve essere **AX** (word) oppure **AL** (byte).
Questa istruzione consente di trasferire una word o un byte per volta. Se la porta ha indirizzo = 255 si può usare l'indirizzamento diretto:

```
IN AL,01AH
IN AX,080H
```

Se la porta ha indirizzo >255 si deve usare l'indirizzamento tramite DX

```
MOV DX,8000H
IN AL,DX
IN AX,DX
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 583

Comunicazione con dispositivi I/O

Esegue i trasferimenti dati dal processore verso lo spazio di I/O.

OUT ind_porta_IO, registro

dove il registro deve essere **AX** (word) oppure **AL** (byte).
Questa istruzione consente di trasferire una word o un byte per volta. Se la porta ha indirizzo = 255 si può usare l'indirizzamento diretto:

```
OUT 0FFH, AL
OUT 0FFH, AX
```

Se la porta ha indirizzo >255 si deve usare l'indirizzamento tramite DX

```
MOV DX,0400H
OUT DX,AL
OUT DX,AX
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 584

Comunicazione con dispositivi I/O

Esempio : Comandare (accendere e spegnere) una periferica utilizzando la porta di I/O di indirizzo 0FFH.

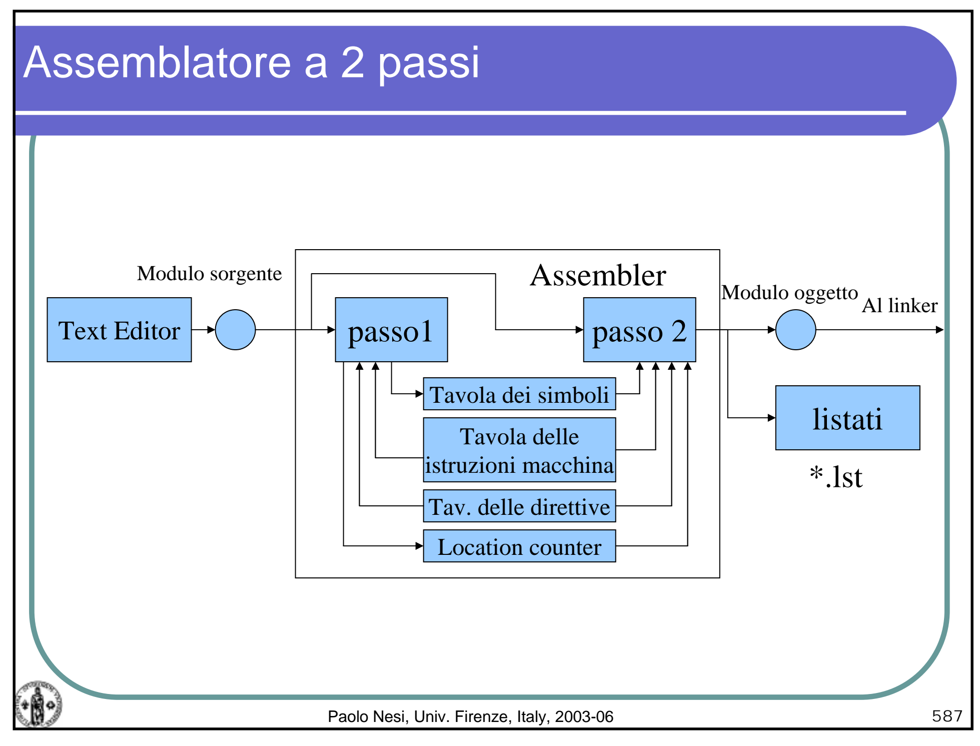
```
PORTA EQU 0FFH   ; assegna FF a COMMAND
MOV AL, 01H               ;
OUT PORTA, AL             ; accende la periferica
…
MOV AL, 0H                ;
OUT PORTA, AL             ; spegne la periferica
```

Scrivendo un “1” per mezzo della OUT si attivano sia la decodifica dell'indirizzo che il segnale IOWR# la cui contemporaneità causa l'accesso alla periferica.

Per lo spegnimento basta caricare uno zero in AL prima di eseguire la OUT. Il segnale di comando per la periferica sarà modificato solo dagli accessi alla porta di indirizzo FFH.

Paolo Nesi, Univ. Firenze, Italy, 2003-06 585

Calcolatori Elettronici
CDL in Ingegneria Elettronica
Facolta' di Ingegneria,
Universita' degli Studi di Firenze
Nuovo Ordinamento
Parte 11: Esempi di Programmazione
Dr. Ing. Ivan Bruno
Prof. Paolo Nesi
http://www.dsi.unifi.it/~ivanb
ivanb@dsi.unifi.it
AA 2005-2006
Paolo Nesi, Univ. Firenze, Italy, 2003-06
586


Assemblatore a 2 passi
Modulo sorgente
Text Editor
Assembler
passo1
passo 2
Modulo oggetto
Al linker
Tavola dei simboli
Tavola delle istruzioni macchina
Tav. delle direttive
Location counter
listati
*.lst
Paolo Nesi, Univ. Firenze, Italy, 2003-06
587

Assemblatore Passo 1

- Fornire le locazioni degli identificatori (variabili e label)
- La locazione degli identificatori avviene per mezzo di una variabile detta Location Counter (LC)
  - Ad ogni istruzione incontrata il valore del LC viene incrementato di una q.tà pari al numero di byte richiesti per codificarla
  - Ad ogni direttiva di allocazione in memoria il valore viene incrementato del numero di byte richiesti dalla direttiva
- LC viene azzerato nel passare da un segmento ad un altro
- E' usato per costruire la Tavola dei Simboli

Tavola dei Simboli

```
DATA_SEG        SEGMENT                                     ;(LC)=0  NBA=0
    ORG 10                                                  ;(LC)=0  NBA=10
    ARRAY       DB          -29                             ;(LC)=10NBA=1
    RESULT                  DW       15                     ;(LC)=11NBA=2
    TOT         DB          82H,04H,2AH                     ;(LC)=13 NBA=3
    …
DATA_SEG ENDS
CODE_SEG        SEGMENT                                     ;(LC)=0  LB=0
    ASSUME CS:CODE_SEG, DS:DATA_SEG                         ;(LC)=0  LB=0
    START:      MOV AX, DATA_SEG                            ;(LC)=0  LB=3
                MOV DS, AX                                  ;(LC)=3  LB=2
                MOV AX, RESULT                              ;(LC)=5  LB=4
                …
CODE_SEG ENDS                                   ;NBA = N° DI BYTE
          END START                             ;LB = LUNGHEZZA IN BYTE
```

Tavola dei Simboli

SIMBOLO	OFFSET	SEGMENTO	TIPO
DATA_SEG	00H		SEGMENT
ARRAY	0AH	DATA_SEG	BYTE VARIABLE
RESULT	0BH	DATA_SEG	WORD VARIABLE
TOT	0DH	DATA_SEG	BYTE VARIABLE
…			
CODE_SEG	00H		SEGMENT
START	00H	CODE_SEG	NEAR LABEL
…			

Paolo Nesi, Univ. Firenze, Italy, 2003-06 590

Tavola dei Simboli - Passo 1

- Una variabile o label è DEFINITA quando questa compare nel primo campo a sinistra di una dichiarazione nel programma sorgente
- Se quando viene incontrata è già presente nella Tavola dei Simboli si genera un errore, altrimenti viene aggiunta nella tavola
- Si ha errore quando uno mnemonico di istruzione o di direttiva non compare nella Tavola dei Simboli Permanenti (istruzioni e direttive)
- Il passo 1 termina quando si incontra la direttiva END

Paolo Nesi, Univ. Firenze, Italy, 2003-06 591

Tavola dei Simboli - Passo 2

- Per mezzo della tavola dei simboli si assemblano le istruzioni e sulla base degli operandi (indirizzamenti) vengono scelte le codifiche in codice macchina sfruttando la Tavola dei simboli permanenti contenente
  - **Tavola delle istruzioni macchina**
  - **Tavola delle direttive**
- Le costanti pre-assegnate e incontrate nelle dichiarazioni dei dati vengono inserite
- Le espressioni fornite come operandi vengono quindi valutate e sostituite con il valore risultante
- Vengono calcolati gli offsets da associare

Paolo Nesi, Univ. Firenze, Italy, 2003-06 592

Osservazioni

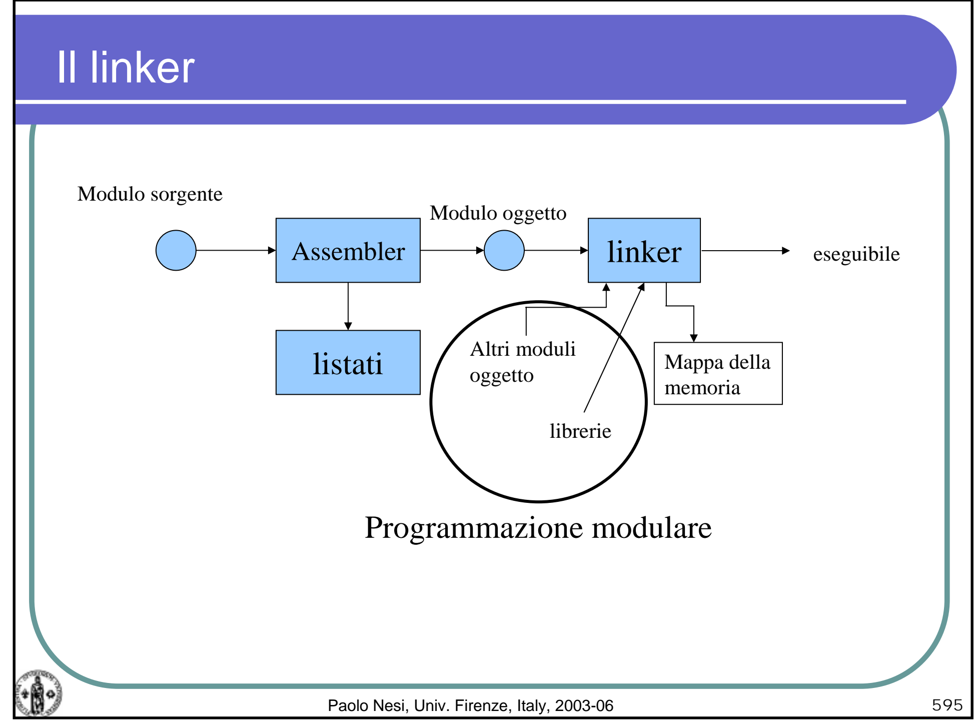
- Quando l'assemblatore al passo 1 incontra in un operando una variabile o label
  - questa può già trovarsi nella Tabella dei Simboli (BACKWARD REFERENCE)
  - Se la variabile o la label non compare nella Tavola dei Simboli vuol dire che ancora non è stata incontrata una direttiva che l'ha definita e che forse più avanti nel listato lo sarà (FORWARD REFERENCE).
- Il caso di forward reference è una situazione critica per l'assemblatore:
  - il problema consiste nell'incremento del location counter in quanto non si conosce ancora il tipo.
- L'assemblatore può supporre una dimensione di default,
  - essa potrà risultare in difetto oppure in eccesso rispetto alla reale dimensione della variabile o della label.
  - Nel primo caso si genera un errore di assemblaggio,
  - Nel secondo si deve completare lo spazio in eccesso con delle istruzioni NOP (Nessun operazione)

Paolo Nesi, Univ. Firenze, Italy, 2003-06 593

Osservazioni

- L'inserimento di NOP rallenta l'esecuzione
- Per ovviare al problema delle forward reference una soluzione è:
  - esplicitare il tipo della variabile o label usando l'operatore PTR
    - BYTE PTR, WORD PTR o DWORD PTR per le variabili
    - NEAR PTR o FAR PTR per le label
- Si ha forward reference anche quando si usa una variabile che è definita in un segmento con registro di segmento diverso da quello di default: in questo caso si assegna il DS corrente come default e si assembla l'istruzione.
- Quest'ultimo caso può essere risolto esplicitando nell'istruzione il registro di segmento cui appartiene la variabile (es. ES:VAR) oppure settare momentaneamente il DS al segmento per quella variabile e poi ripristinare il precedente valore di DS

Il linker



Programmazione modulare (cenni)

- Il programma principale chiama un sottoprogramma con una call Far
  - **EXTRN <subprogram name>: type**
  - Type può essere FAR, NEAR
    - Dice all'assemblatore che c'è un sottoprogramma che si trova in un segmento diverso.
  - **PUBLIC <subprogram name>**
    - Dice all'assemblatore E al linker che il sottoprogramma definito nello specifico segmento deve essere disponibile agli altri moduli che provvederanno a definirla EXTRN.

Paolo Nesi, Univ. Firenze, Italy, 2003-06

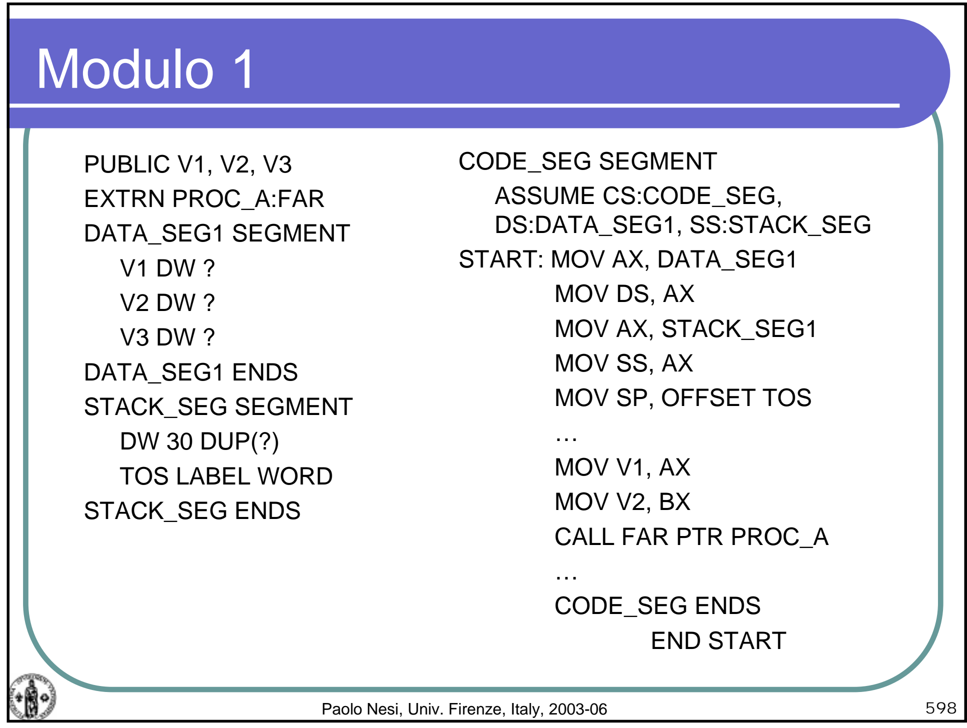
596

Programmazione modulare (cenni)

- Per le variabili
  - **EXTRN <variabile>: type**
  - Type può essere BYTE, WORD o DWORD
    - Dice all'assemblatore che ci sono delle varibili definite in un segmento diverso.
  - **PUBLIC <variabile>**
    - Dice all'assemblatore e al linker che la variabile definita nello specifico segmento deve essere disponibile agli altri moduli che provvederanno a definirla EXTRN.

Paolo Nesi, Univ. Firenze, Italy, 2003-06

597



Modulo 2

```
EXTRN V1:WORD, V2:WORD, V3:WORD
PUBLIC PROC_A
CODE_SEG1 SEGMENT
    ASSUME CS:CODE_SEG1
    PROC_A PROC FAR
        PUSH AX
        MOV AX, V1
        ADD AX, V2
        MOV AX, V3
        POP AX
        RET
    PROC_A ENDP
CODE_SEG1 ENDS
        END
```

Paolo Nesi, Univ. Firenze, Italy, 2003-06 599

Calcolatori Elettronici
Fine del corso
Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
AA 2005-2006
Paolo Nesi, Univ. Firenze, Italy, 2003-06
600