Facoltà di Ingegneria, Università degli Studi di Firenze

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Nuovo Ordinamento

Slide del corso

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi, nesi@dsi.unifi.it
AA 2006-2007
Ver:6.6

1

Calcolatori Elettronici, CDL Nu.ord.

- Parte 0: Introduzione
- Parte 1: L'algebra di Boole e i sistemi di numerazione
- Parte 2: Logica Combinatoria
- Parte 3: Logica Sequenziale
- Parte 4: Le architetture degli elaboratori
- Parte 5: L'architettura software e la scelta
- Parte 6: La CPU 8086
- Parte 7: Il Sistema di I/O
- Parte 8: Altri Aspetti ed evoluzione
- Parte 8bis: La programmazione
- Parte 9: La programmazione in Assembly, le istruzioni
- Parte 10: Procedure, Stack ed Interruzioni
- Parte 11: Esempi di Programmazione

Suggerimenti

- Le dispense/slide sono la traccia della lezione ma non sono il libro, si studia sul libro, e anche su altro materiale
- Potete accedere ad un sito web sul quale trovate tutto il materiale: http://www.dsi.unifi.it/~nesi
  - Regole per l'esame e compitini, etc.
  - Dispense, slide, etc.
  - Riferimenti bibliografici: lista libri consigliati, etc.
  - Strumenti software: simulatore di logica, etc.
  - Testi di Vecchi Compiti e Compitini
  - Esercizi di Assembly, Assemblatore, etc.
  - etc.
- Le slide saranno aggiornate e messe sul sito web in formato PDF via via che saranno pronte
- Durante il corso vengono organizzate delle esercitazioni durante le ore di lezione con esercizi del tutto simili a quelli dei compiti/compitini.

Modalità di esame

- L'esame e' solo scritto, un compito
  - Risposte aperte e chiuse
- Viene tipicamente organizzato un compitino a meta' corso
  - Le tematiche fanno riferimento a circa la meta' dei contenuti, inciderà per circa la meta' del voto finale, una stima precisa viene fatta in base al testo dello stesso.
  - Tutti possono partecipare: studenti in corso e/o degli anni passati e/o fuori corso, etc.
  - Supera il compitino solo chi prende un voto >= 18/30
- A fine corso lo Studente puo' scegliere di fare:
  - il secondo compitino sulla seconda parte del corso, solo per chi ha superato il primo compitino (il voto viene stimato come media pesata dei voti dei due compitini), oppure
  - Il compito completo su tutto il programma del corso (il voto dipende solo da come viene svolto il compito completo)

Principale Libro di Testo

- G. Bucci
- McGrawHill

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 0, Introduzione

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2007

Cenni Storici

- Greci e Romani avevano gia' delle prime realizzazioni di macchine per automatizzare il calcolo
- 1642 appare la macchina di Biagio Pascal
- 1671 Leibniz presenta la prima macchina per le moltiplicazioni tramite addizioni successive
- 1823 Charles Babbage (1792-1871) presenta la prima macchina da calcolo di utilizzo generale senza pero poterla completare. La meccanica del periodo non era sufficientemente sofisticata per lo scopo.
- 1946 ENIAC, la prima macchina da calcolo a valvole
- 1948 IBM 604, calcolatore a valvole/tubi
- 1951 UNIVAC, calcolatore a valvole/tubi
- 1971 INTEL 4004, il primo C*alcolatore Elettronico*
  - capace di lavorare con *numeri di 4 bit* …… ma cosa sono i bit ?

(a questo punto e' necessario andare a vedere cosa e' un calcolatore)

Cenni Storici

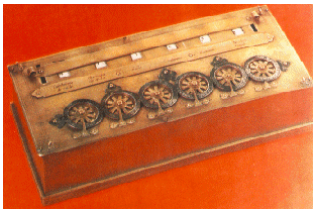
1642 -- Macchina calcolatrice di Pascal

•Dimensioni di una scatola per scarpe

•interazione con delle "rotelline" sulle quali erano riportati i numeri da 1 a 9.

•La novità rivoluzionaria: introduzione del principio del "riporto automatico".

•L'unico limite dell'invenzione di Pascal era che la macchina permetteva soltanto di eseguire addizioni e sottrazioni.



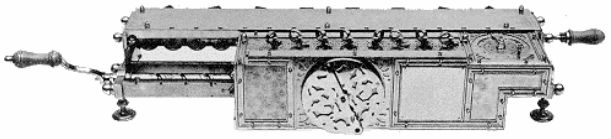
Cenni Storici(2)

1671 -- Modifiche apportate alla macchina calcolatrice da Leibniz

•Perfezionò la "calcolatrice" di Pascal inventando una macchina in grado di eseguire anche moltiplicazioni e divisioni.

•Erano macchine non programmabili ma dotate di programmi definiti dalla meccanica stessa della macchina.

•Fondatore del **sistema di numerazione binario** su cui si basa il funzionamento di tutti i computer moderni.



Paolo Nesi, Univ. Firenze, Italy, 2003-07 9

Cenni Storici (3)

1801 -- Jacquart inventa la scheda perforata

Invenzione di un telaio per la tessitura automatica che funzionava per mezzo di una serie di schede perforate dove, la posizione dei fori guidava il filo a formare un certo disegno nell'ordito.

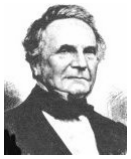
Le schede erano la prima forma di programmazione di una macchina automatica.

Paolo Nesi, Univ. Firenze, Italy, 2003-07 10

Cenni Storici (4)

1823 -- Macchina di Babbage

Sfruttava il principio sia della macchina di Pascal sia del telaio di Jacquard.
Due serie di schede perforate: una serie costituiva il programma (le istruzioni), l'altra serie di schede rappresentavano i dati (i valori).
Viene considerato il "Padre dei computer", tanto che le sue idee sono alla base della moderna programmazione.



Introduzione alla programmazione

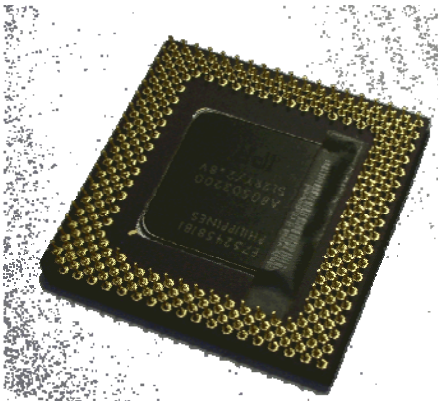
- Il calcolatore elettronico è uno strumento in grado di eseguire insiemi di *azioni* ("*mosse*") *elementari*
- le azioni vengono eseguite su oggetti (***dati***) per produrre altri oggetti (***risultati***)
- l'esecuzione di azioni viene richiesta all'elaboratore attraverso *frasi* scritte in qualche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

Programmazione

- È l'attività con cui si predispone l'elaboratore a **eseguire un *particolare insieme di azioni* su *particolari dati***, allo scopo di *risolvere un problema.*

Scheda Madre

CPU



Paolo Nesi, Univ. Firenze, Italy, 2003-07 15

Evoluzione dei calcolatori

Data di introduzione	Nome del chip	N. di transistori (/1000)	Tecnologia (μm)	Frequenza (MHz)
Novembre 1971	4004	2,3	10	0,108
Aprile 1972	8008	3,5	10	0,500
Aprile 1974	8080	4,5	6	2
Giugno 1978	8086	29	3	5
Febbraio 1982	80286	134	1,5	8
Ottobre 1985	80386	275	1,5	16
Aprile 1989	80486	1.200	1	25
Marzo 1993	Pentium	3.100	0,8	60
Novembre 1995	PentiumPro	5.500	0,6	150
Maggio 1997	Pentium II	7.500	0,35	233
Febbraio 1999	Pentium III	9.500	0,25	450
Novembre 2000	Pentium 4	42.000	0,18	1400

Tabella 1.2 Aumento del numero di transistori delle CPU Intel. I dati riportati si riferiscono al modello di introduzione. Per i modelli introdotti in più versioni, la tabella riporta i dati relativi alla versione di più bassa capacità. Per esempio, il PentiumPro è stato introdotto in ben quattro versioni, di cui la meno potente (quella riportata) era tecnologia a 0,6 μm e frequenza pari a 150 MHz, mentre la più avanzata era in tecnologia a 0,35 μm e frequenza pari a 200 MHz.

16

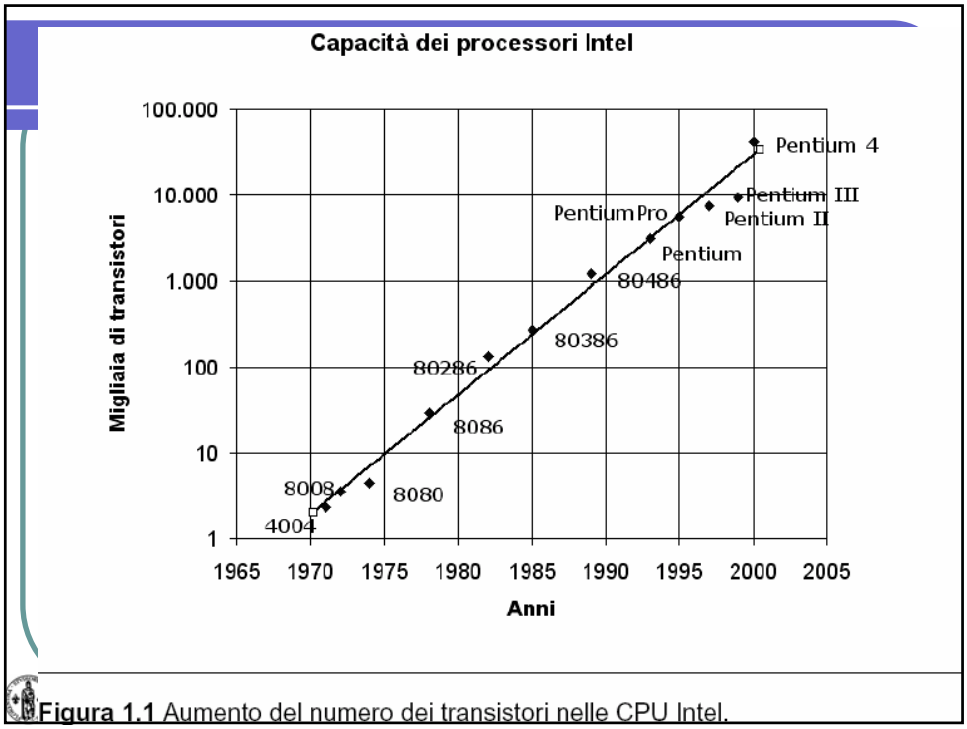


Figura 1.1 Aumento del numero dei transistori nelle CPU Intel.

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 1a, L'Algebra di Boole

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2007

ALGEBRA di BOOLE

- George Boole matematico inglese (XIX secolo)
- La sua algebra viene utilizzata solo dall'inizio del XX secolo (primi sistemi di calcolo)
- Si basa su due soli valori:
  - acceso (ON, vero, alto, true..)
  - spento (OFF, falso, basso, false, …)

Variabili e Proposizioni

- Le variabili possono assumere solo due valori:
  - 0 L F
  - 1 H T
- Si chiamano *Variabili logiche* o *Booleane*
- *Sono proposizioni le espressioni dell'algebra Booleana con certi operatori:*

A or B and C

Costanti Booleane

- Oltre alle variabili vi sono anche le costanti
- Essendo l'Algebra Booleana definita su due soli simboli, esistono solo due possibili costanti:
  - 0 Falso basso….
  - 1 Vero alto….

Funzioni Booleane

- Usando le variabili Booleane, si possono costruire le funzioni Booleane

$$F(x,y,z)$$

- Funzioni che possono assumere solo due valori: 0,1
- Variabili Booleane e non:
  - F(x,y,z) = X and Y or Z
  - F(x,y,z) = X > (Y + Z)

Operatori

- Esistono due tipi di operatori, in relazione al numero di variabili che utilizzano:
  - Monadici, detti anche unari
  - Diadici, detti anche binari
- Questi possono essere
  - Logici
  - Confronto

Operatori di Confronto

- Maggiore: >
- Maggiore uguale: >=, ≥, ….
- Minore: <
- Minore uguale: <=, ≤, ….
- Diverso: !=, <>, ≠, …
- Uguale: ==, =, ……

Il loro risultato e' un numero Booleano

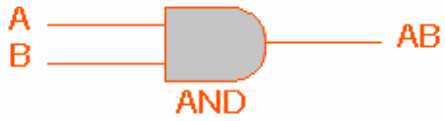
Operatori Booleani

- Congiunzione: AND, e, &, ∧, &&, ….
- Disgiunzione: OR, oppure, |, V, ||, +, ….
- Negazione o complemento:
  - NOT, not, no, !, -, *, ~, ¬, ….
  - Negato (A negato): $\overline{A}$, not A, *A, …..
- Implica: →
- Co-implica: ←→

L'operatore AND

- Il risultato è *vero* solo se sono *vere* entrambe le variabili

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1



Paolo Nesi, Univ. Firenze, Italy, 2003-07 27

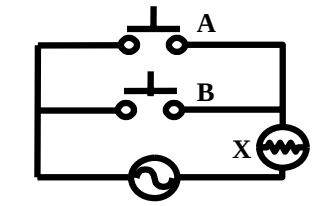
L'operatore OR

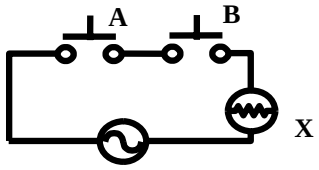
- Il risultato è *vero* solo se è *vera* almeno una delle variabili

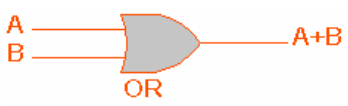
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

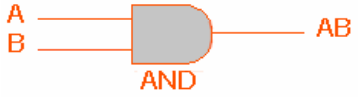
Paolo Nesi, Univ. Firenze, Italy, 2003-07 28

Similitudine con i circuiti elettrici





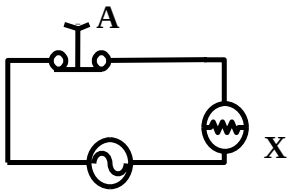


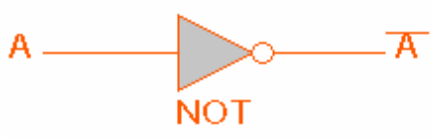


L'operatore NOT

- Il risultato è il complemento

A	NOT A
0	1
1	0





Operatore Implica

- A → B che si legge A IMPLICA B
- Il termine **implica** non B e' vero (o falso) come potrebbe lasciare intuire il significato comune del termine implica.

A	B	-A	-A or B
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

L'operatore Coimplica

- A <-> B
- che si legge A COIMPLICA B e' equivalente a A -> B and B->A

A	B	-A	-B	-A or B	-B or A	-AorB and –BorA
0	0	1	1	1	1	1
0	1	1	0	1	0	0
1	0	0	1	0	1	0
1	1	0	0	1	1	1

Proprietà degli operatori

OR	AND
X v 1 = 1	X ^ 1 = X
X v 0 = X	X ^ 0 = 0
X v X = X	X ^ X = X
X v –X = 1	X ^ -X =0

Precedenze tra operatori

- Le precedenze sono simili alla somma ‘+’ e al prodotto ‘x’ dell’algebra numerica consueta:
  - priorità alta, maggiore x
  - priorità bassa, minore +
  - Si possono inoltre utilizzare tipicamente le parentesi tonde, ( ), per forzare la precedenza

Proprietà dell'algebra Booleana

$X \cdot 0 = 0$	$X + 1 = 1$	
$X \cdot 1 = X$	$X + 0 = X$	
$X \cdot X = X$	$X + X = X$	idempotenza
$X \cdot \overline{X} = 0$	$X + \overline{X} = 1$	complementazione
$X \cdot Y = Y \cdot X$	$X + Y = Y + X$	commutativa
$X \cdot (X + Y) = X$	$X + (X \cdot Y) = X$	assorbimento
$X \cdot (\overline{X} + Y) = X \cdot Y$	$X + (\overline{X} \cdot Y) = X + Y$	assorbimento
$X \cdot (Y + Z) = X \cdot Y + X \cdot Z$		
$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$		distributiva

Proprietà dell'algebra Booleana

$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z = X \cdot Y \cdot Z$
$X + (Y + Z) = (X + Y) + Z = X + Y + Z$ — associativa

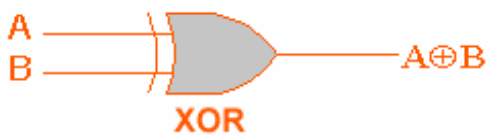
$\overline{(\overline{X})} = X$

$\overline{X \cdot Y} = \overline{X} + \overline{Y}$ $\qquad$ $\overline{X + Y} = \overline{X} \cdot \overline{Y}$ — De Morgan

L'operatore XOR

- Il risultato è *vero* solo se è *vera* solo una delle due variabili

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



Espressioni logiche

- Un insieme di variabili e/o costanti Booleane a cui siano applicati gli operatori logici si dice *espressione Booleana* o *logica*
- Una espressione logica rappresenta una funzione logica, ad esempio:

$$T = a \cdot \overline{b} + \overline{a} \cdot b$$

Funzioni Booleane

- **Funzioni completamente specificate**: se per tutte le combinazioni delle variabili il suo valore è determinato
- Esempio: puoi fare la frittata se hai
  - le uova,
  - una padella,
  - olio
  - sale

Esercizi - Semplificare le seguenti espressioni logiche

1) WXYZ' + WXY'Z' + WX'YZ'+ WX'Y'Z'
(WZ')

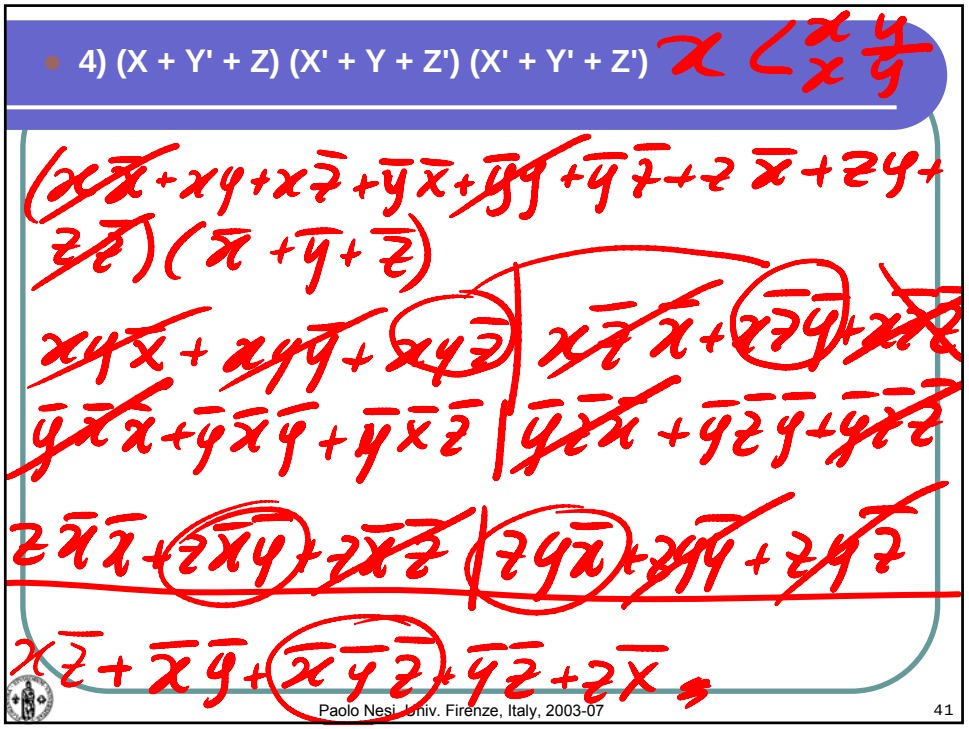
2) (A + B'C)'
(A'(B + C'))

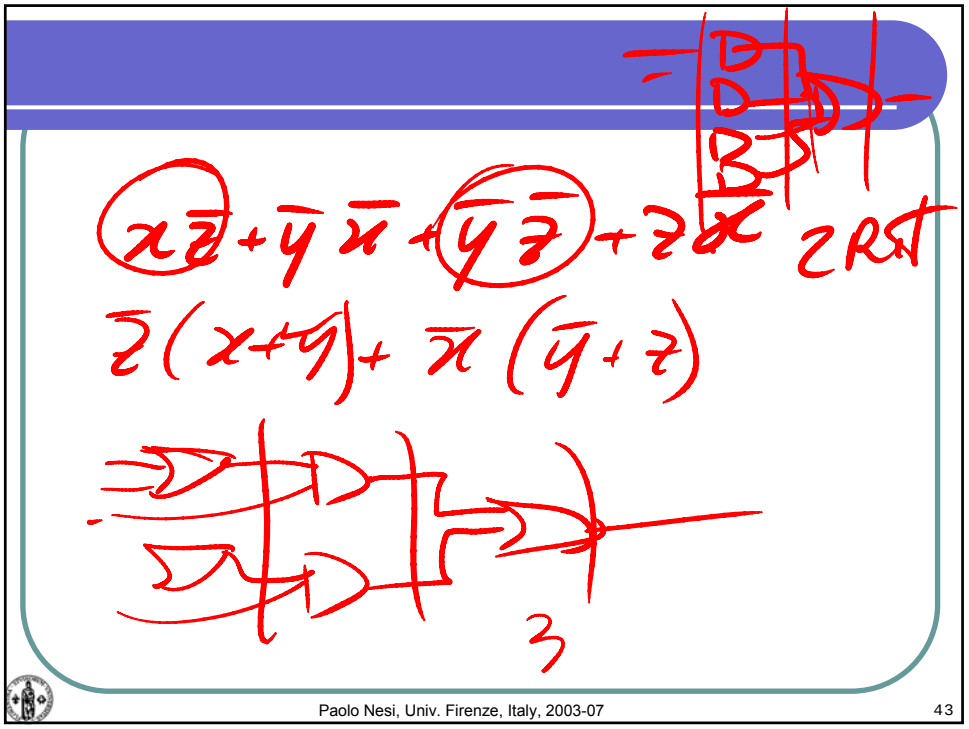
3) ABC + AB'C + ABC'
(A(C + B))

4) (X + Y' + Z) (X' + Y + Z') (X' + Y' + Z')
(XZ' + X'Y' + Y'Z' + X'Z)

A' = Ā

- 4) (X + Y' + Z) (X' + Y + Z') (X' + Y' + Z')





Esercizi - Semplificare le seguenti espressioni logiche

5) A'B'C' + A'B'C + A'BC' + AB'C' + ABC'

A' = Ā

(A'B' + C')

6) A'B'C'D' + A'BC'D' + A'BCD + ABC'D' + ABC'D + ABCD + ABCD' + AB'C'D'

(C'D' +BCD + AB)

7) ABC + C (AB(D + A'(C'D + B) + B'D) + AB') + AB'C'

(AC + AB')

8) ((AB') (CD))' ((A'BC) (BD))' ((BC') (A + B + C))'

(A'B' + ABC + CD' + B'C' + B'D')

A' = Ā

Paolo Nesi, Univ. Firenze, Italy, 2003-07 44

Funzioni Booleane

- **Funzioni non completamente specificate**: se a una o più combinazioni delle sue variabili non corrisponde alcun valore della funzione (dont' care)
- Esempio: puoi fare la frittata se hai
  - le uova,
  - una padella,
  - Olio
  - ---

Tabella della verità

- Ogni funzione Booleana è caratterizzata dalla propria *tabella della verità*

x	y	z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Operatori NAND/NOR

- Con gli operatori NOT, OR, AND, XOR si possono costruire tutte le funzioni Booleane
- Esistono due operatori (NAND, NOR) che permettono la sintesi di *qualsiasi funzione*, utilizzando un unico tipo di operatori

L'operatore NAND

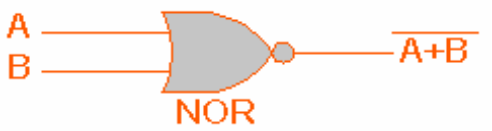
- Il risultato è *vero* solo se è *falso* l'AND tra le due variabili

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

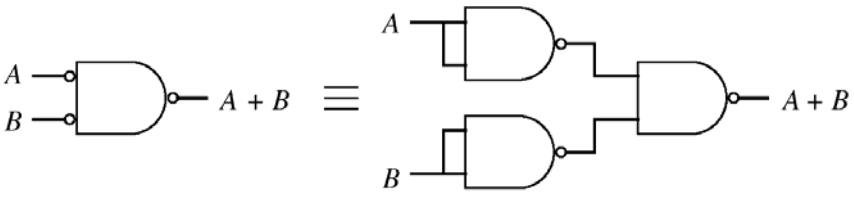
L'operatore NOR

- Il risultato è *vero* solo se è *falso* l'OR tra le due variabili

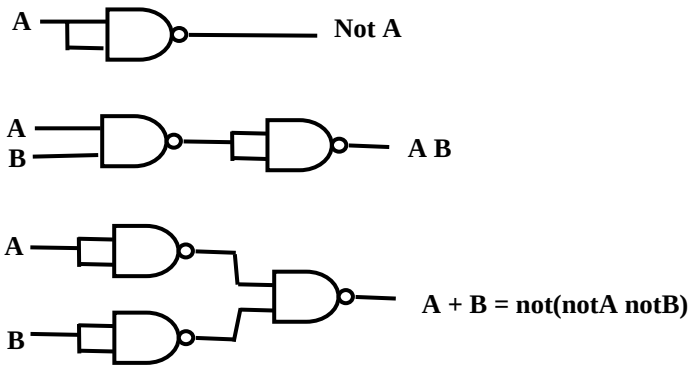
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



Tutto con NAND (NOR)



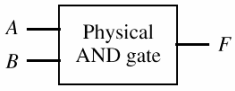
Tutto con i NAND a 2



Logica Positiva e Negativa

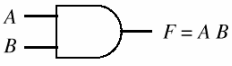
Voltage Levels

A	B	F
low	low	low
low	high	low
high	low	low
high	high	high



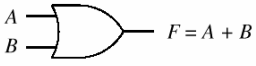
Positive Logic Levels

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



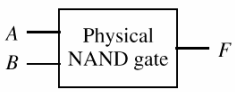
Negative Logic Levels

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0



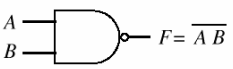
Voltage Levels

A	B	F
low	low	high
low	high	high
high	low	high
high	high	low



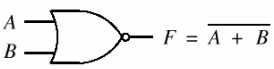
Positive Logic Levels

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



Negative Logic Levels

A	B	F
1	1	0
1	0	0
0	1	0
0	0	1



Proprietà operatori NAND/NOR

NOR (↑)	NAND (↓)
$X \uparrow 1 = 0$	$X \downarrow 0 = 1$
$X \uparrow 0 = \bar{X}$	$X \downarrow 1 = \bar{X}$
$X \uparrow X = \bar{X}$	$X \downarrow X = \bar{X}$
$\overline{\bar{X} \uparrow \bar{Y}} = X \cdot Y$	$\overline{\bar{X} \downarrow \bar{Y}} = X + Y$
$\overline{X \uparrow Y} = X + Y$	$\overline{X \downarrow Y} = X \cdot Y$

Paolo Nesi, Univ. Firenze, Italy, 2003-07 53

Data sheet: NAND x 2

SN7400 QUADRUPLE 2-INPUT POSITIVE-NAND GATES

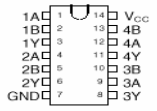
description

These devices contain four independent 2-input NAND gates.

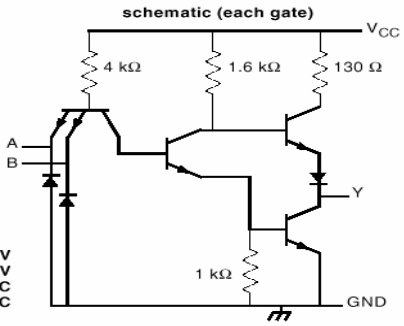
function table (each gate)

INPUTS A	INPUTS B	OUTPUT Y
H	H	L
L	X	H
X	L	H

package (top view)



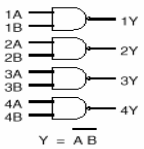
schematic (each gate)



absolute maximum ratings

Supply voltage, VCC	7 V
Input voltage:	5.5 V
Operating free-air temperature range:	0 °C to 70 °C
Storage temperature range	– 65 °C to 150 °C

logic diagram (positive logic)



$Y = \overline{A B}$

recommended operating conditions

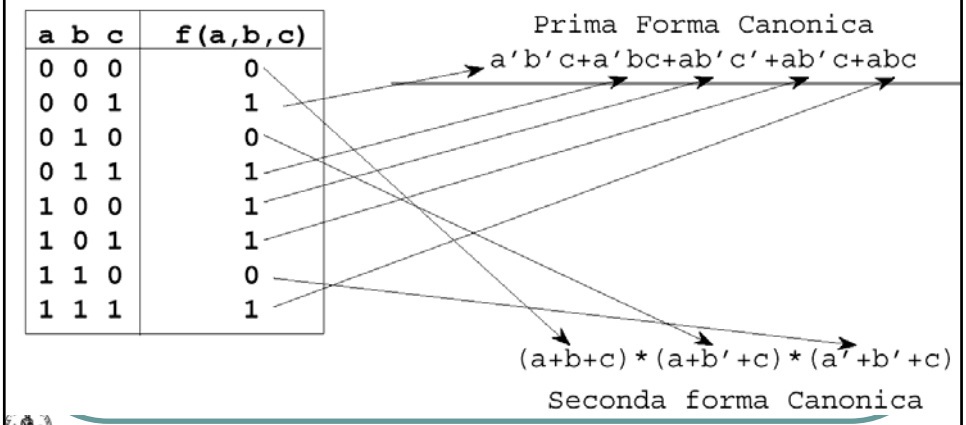
		MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage	4.75	5	5.25	V
V_{IH}	High-level input voltage	2			V
V_{IL}	Low-level input voltage			0.8	V
I_{OH}	High-level output current			– 0.4	mA
I_{OL}	Low-level output current			16	mA
T_A	Operating free-air temperature	0		70	°C

54

Forme Canoniche

- Prodotti di Somme
- Somme di Prodotti

a	b	c	f(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1



Sintesi Somme di Prodotti

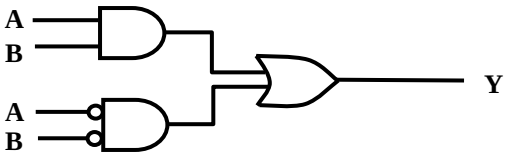
- Si identifica i termini che danno 1
  - A=0 allora not A
  - A=1 allora A
- Y = not A and not B + A and B

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Circuito: somma di prodotti

- Y = not A and not B + A and B
- Produrre la rete

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1



Sintesi Prodotti di Somme

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

- Si identifica i termini che danno 0, logica negata
  - A=0 allora A
  - A=1 allora not A
- Y = (not A + B) and (A + not B) =
  - (not A + B) and (A + not B) = not A and A + not A and not B + B and A + B and not B
- in base alle proprietà di invarianza: not A and A = 0, B and not B= 0, pertanto:
  - (not A + B) and (A + not B) = not A and not B + A and B

Riduzione/semplificazione

A	B	C	U
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	/
1	1	0	/
1	1	1	/

In termini di somme di prodotti si ha

A'B'C + A'BC + AB'C' (1)

Semplificato diventa

A'C + AB'C' (2)

Ma se si considerano anche le combinazioni degli ingressi che non vengono proposte in ingresso:

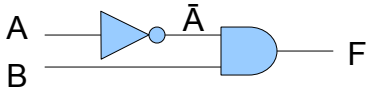
AB'C + ABC' + ABC

Si ha che la (2) si puoà ancora semplificare ottenendo:

A'C + AB'

Sintesi circuitale di espressione logiche

- F= ĀB



A	B	F
0	0	0
0	1	1
1	0	0
1	1	0

- F = A(C+B)

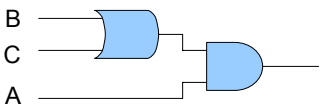


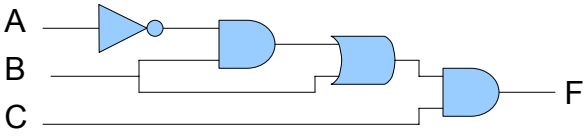
Tabella verità?

- F = (A'B' + ABC + CD' + B'C' + B'D') ? ...

Circuito + Tabella verità?

Sintesi circuitale di espressione logiche

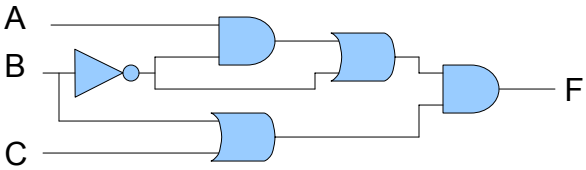
- F= ? Tabella della verità?



A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$$F = C(B+(B\bar{A}))$$

Esercizi



- Calcolare F e scrivere la tabella della verità.

Operazioni Binarie su sequenze di bit

- AND sui bit

  101101 and
  010110 =

  000100

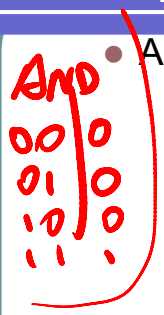
- OR sui bit
  - …
- NOT sui bit
  - ….

Operazioni logiche sui bit

- AND sui bit

  10101010
  01011101

  00001000



- XOR sui bit

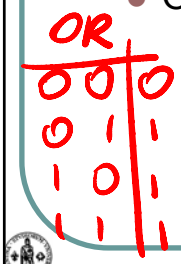
  10101010
  01011101

  11110111

- OR sui bit

  10101010
  01011101

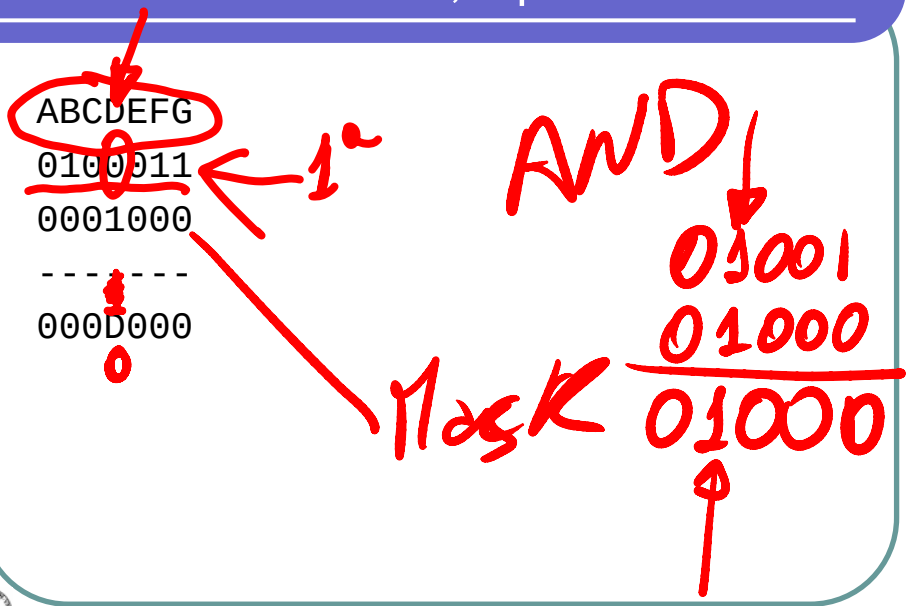
  11111111



- NOT sui bit

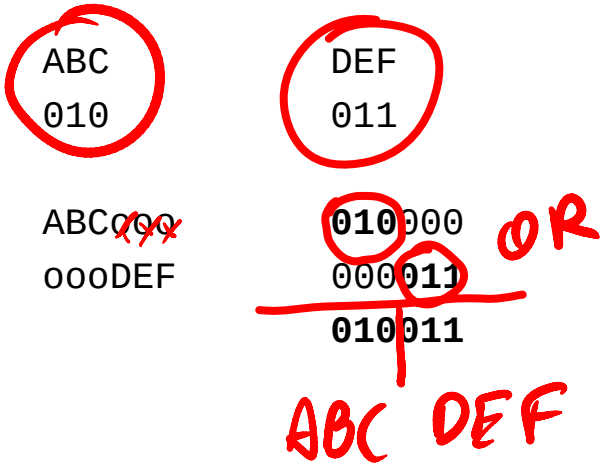
  10101010 ⇒ 01010101

Mascheratura sui bit, Operatore AND



Paolo Nesi, Univ. Firenze, Italy, 2003-07 66

Combinazione sui bit, operatore OR



Paolo Nesi, Univ. Firenze, Italy, 2003-07 67

Bit, Nibble, byte, word ……

Un bit rappresenta una cifra binaria.

Il bit però è un'unità di informazione troppo piccola per poter essere elaborata in modo efficiente.

I bit pertanto sono trattati secondo i seguenti gruppi:

1 nibble = 4 bit
1 byte = 8 bit
1 word = 16 bit
1 doubleword = 32 bit
1 Kilobyte = 2^{10} byte = 1024 byte = 8196 bit
1 Megabyte = 2^{20} byte = 1048576 byte ~ 8 milioni di bit
1 Gigabyte = 2^{30} byte ~ 1 miliardo di byte ~ 8 miliardi di bit
1 Terabyte = 2^{40} byte ~ 10^{12} byte ~ 2^{43} bit
1 Exabyte = 2^{50} byte

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 1b, I Sistemi di Numerazione

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2007

Rappresentazione dell'Informazione

In un calcolatore si possono rappresentare vari tipi di informazioni:

- Numeri reali
- Numeri interi
- Testi
- Grafici
- ….
- Disegni
- Fotografie
- Filmati
- Suoni
- …

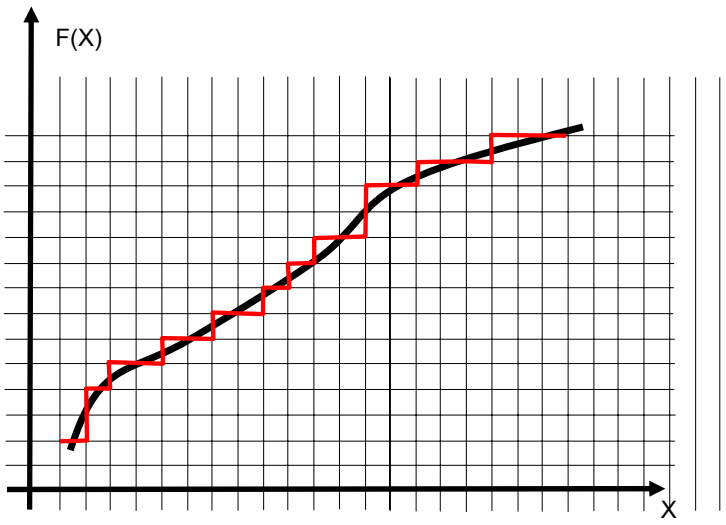
Rappresentazione dell'Informazione

L'informazione può essere rappresentata in due forme:

- **Analogica**
  - la grandezza è rappresentata in modo continuo.
- **Digitale**
  - una grandezza è rappresentata in modo discreto.

•Gran parte delle grandezze fisiche sono di tipo continuo (ad esempio un segnale acustico).
•Tuttavia alcuni tipi di informazioni "artificiali" sono di tipo discreto (ad esempio un testo scritto).

Da analogico a digitale



Rappresentazione dell'Informazione

- Per elaborare delle grandezze di tipo continuo con un calcolatore, bisogna utilizzare/ trasformarle in una rappresentazione digitale, cioè discreta.
- La rappresentazione digitale è una approssimazione della rappresentazione analogica.
- L'errore di approssimazione dipende dalla precisione della rappresentazione digitale utilizzata per la codifica interna nel calcolatore.

Rappresentazione dell'Informazione

- Internamente ad un elaboratore ogni informazione è rappresentata da una sequenza di bit (cifre binarie)
- Una sequenza di bit può rappresentare entità diverse. Il modello di rappresentazione da la chiave di lettura.
- Ad esempio la sequenza di cifre binarie 01000001 può rappresentare:
  - l'intero 65
  - il carattere A
  - il valore di un segnale musicale
  - il codice del colore di un punto sullo schermo
  - etc.

Codici

- Un codice è un sistema di simboli atto a rappresentare una informazione di qualsiasi genere (caratteri, numeri, etc.).
- Ogni simbolo è messo in corrispondenza biunivoca con una entità che si vuole rappresentare
- Un codice binario usa come simboli le cifre binarie "0" e "1"

Sistemi di Numerazione Posizionali

- di una base, *b*
- di un insieme ordinato di cifre *d = {……..}*, distinte l'una dall'altra con dimensione pari a quella della base: *dim(d) = b*, L'insieme contiene simboli che hanno una certa posizione nella base ordinata.
- di un codice di interpretazione cioè di un insieme di regole che permettono di determinare quale sia il numero rappresentato da un gruppo di cifre,
- di un insieme di regole e di algoritmi per definire le operazioni fondamentali.

Sistemi di Numerazione Posizionali

- Un numero N può essere rappresentato come una sequenza di cifre:

$$N=d_{n-1}d_{n-2}\ldots\ldots d_1 d_0 \,.\, d_{-1}d_{-2}\ldots\ldots d_{-m}$$

corrispondente a (forma polinomia):

$$N=\underbrace{d_{n-1}\cdot b^{n-1}+d_{n-2}\cdot b^{n-2}+\ldots+d_0\cdot b^{o}}_{\text{PARTE INTERA}}+\underbrace{d_{-1}\cdot b^{-1}+\ldots.+d_{-m}\cdot b^{-m}}_{\text{PARTE FRAZIONARIA}}$$

dove:

- d = cifra
- b = base
- n = numero cifre parte intera
- m = numero cifre parte decimale

Sistema di Numerazione decimale

Sistema Decimale

- Indicato con “10”, “d”, “D”
- b=10, d={0,1,2,3,4,5,6,7,8,9}

$N=d_{n-1}\cdot 10^{n-1}+d_{n-2}\cdot 10^{n-2}+ +d_0+ d_{-1}\cdot 10^{-1}+d_{-m}\cdot 10^{-m}$

- Esempio:

$123.45=1\cdot 10^2+2\cdot 10^1+3\cdot 10^0 +4\cdot 10^{-1}+5\cdot 10^{-2}$

Paolo Nesi, Univ. Firenze, Italy, 2003-07 78

Sistemi di Numerazione Binario

- Indicato con “b”, “2”
- b=2, d={0,1}

$N=d_{n-1}\cdot 2^{n-1}+d_{n-2}\cdot 2^{n-2}+ +d_0\cdot 2^0+ d_{-1}\cdot 2^{-1}+d_{-m}\cdot 2^{-m}$

- Esempio:

$101.01_2 = 1\cdot 2^2+0\cdot 2^1+1\cdot 2^0 +0\cdot 2^{-1}+1\cdot 2^{-2} = 5.25_{10}$

- La cifra binaria è detta “bit” (binary digit – pezzo, pezzetto: cioè l’unità più piccola di informazione).
- E’ il sistema usato nei calcolatori.

Paolo Nesi, Univ. Firenze, Italy, 2003-07 79

2 alla n	N	2 alla - n
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.015625
128	7	0.0078125
256	8	0.00390625
512	9	0.001953125
1024	10	0.0009765625
2048	11	0.00048828125
4096	12	0.000244140625
8192	13	0.0001220703125
16384	14	0.00006103515625
32768	15	0.000030517578125
65536	16	0.0000152587890625
131072	17	0.00000762939453125
262144	18	0.000003814697265625
524288	19	0.0000019073486328125
1048576	20	0.00000095367431640525

80

Sistema di Numerazione ottale

- Indicato con “o”, oppure con il numero “8”
- b=8, d={0,1,2,3,4,5,6,7}

$$N=d_{n-1}\cdot 8^{n-1}+d_{n-2}\cdot 8^{n-2}+ +d_0\cdot 8^0+ d_{-1}\cdot 8^{-1}+d_{-m}\cdot 8^{-m}$$

- Esempio:

$127_8 = 1\cdot 8^2+2\cdot 8^1+7\cdot 8^0 = 87_{10}$

Paolo Nesi, Univ. Firenze, Italy, 2003-07

81

Sistema di Numerazione esadecimale

(Base 16,indicato anche con H, e, 16)

- b=16, d={0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

$N=d_{n-1}\cdot16^{n-1}+d_{n-2}\cdot16^{n-2}+ +d_0\cdot16^0+ d_{-1}\cdot16^{-1}+d_{-m}\cdot16^{-m}$

- Esempio:

$A1_H = A1_{16} = 10\cdot16^1+1\cdot16^0 = 161_{10}$

- Per avere 16 simboli distinti bisogna aggiungere:

$A_H=10$ $C_H=12$ $E_H=14$

$B_H=11$ $D_H=13$ $F_H=15$

Conversioni di Base

Conversione dalla base 10 ad una base qualsiasi: parte intera

$N=d_0\cdot b^0+d_1\cdot b^1+d_2\cdot b^2+ +d_{n-1}\cdot b^{n-1}$

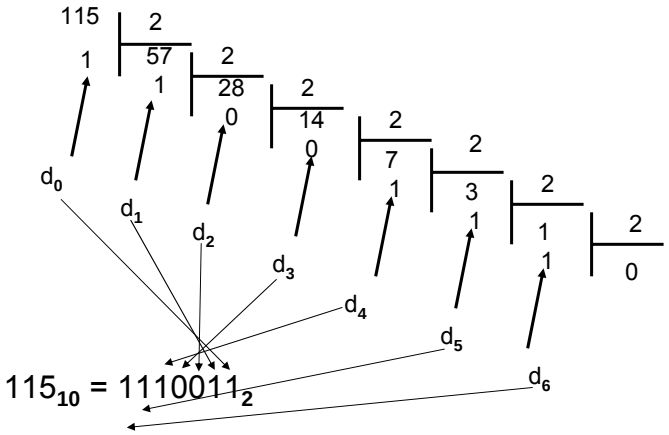
$N=d_0+b(d_1+b(d_2(\ldots +d_{n-1})))$

Quindi dividendo *N* per la base *r* si ottiene come quoziente $d_1+b(d_2(\ldots +d_{n-1}))$ e come (resto d_0 la cifra meno significativa).

Dividendo ancora il quoziente per la base si ottiene la cifra di peso 1 e così via fino ad avere un quoziente 0

Conversioni di Base

Esempio di conversione dalla base 10 alla base 2: parte intera



Conversioni di Base

Conversione dalla base 10 alla base 2: parte frazionaria

$N=d_{-1}\cdot 2^{-1}+d_{-2}\cdot 2^{-2}+ \quad +d_{-m}\cdot 2^{-m}$

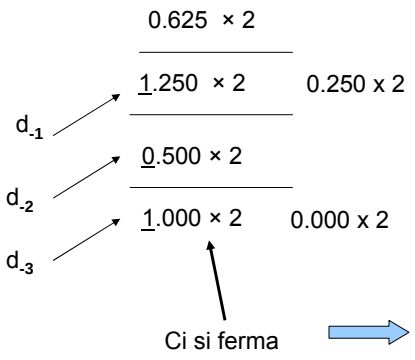
Se si moltiplica N per la base 2 si ottiene per la parte intera d_{-1}.

Moltiplicando ancora per 2 la parte frazionaria si ottiene come parte intera d_{-2} etc.

Il risultato termina quando il risultato della moltiplicazione è esattamente 1 o quando si è raggiunta la precisione voluta (una rappresentazione può essere finita in una base e infinita in un'altra).

Conversioni di Base

Esempio di conversione dalla base 10 alla base 2: parte frazionaria



$0.625_{10} = 0.101_2$

Paolo Nesi, Univ. Firenze, Italy, 2003-07 86

Conversioni di Base

Esempio di conversione dalla base 10 alla base 2: parte intera e frazionaria

Per convertire un numero che ha parte intera e parte frazionaria si effettuano le conversioni separatamente.

Esempio: sia dato il numero decimale 115.625

$$115_{10} = 1110011_2$$
$$0.625_{10} = 0.101_2$$
$$115.625_{10} = 1110011.101_2$$

Paolo Nesi, Univ. Firenze, Italy, 2003-07 87

Conversioni di Base

Conversione fra le basi 8 o 16 e la base 2:

Le notazioni in base 8 e 16 possono essere pensate come delle abbrevviazioni della notazione in base 2.

Ogni cifra ottale corrisponde a 3 cifre binarie:

$101\ 110\ 001_2 = 561_8$

Ogni cifra esadecimale corrisponde a 4 cifre binarie:

$1\ 0111\ 0001_2 = 171_{16}$ $(0001\ 0111\ 0001_2 = 171_{16})$

Tabella Bin, Dec. ed Esadecimale

- Tabella da binario a decimale a esadecimale

Esadecimale	Binario	decimale
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Tabella Decimale a Ottale

- Tabella da binario a ottale a esadecimale

Ottale	Binario	decimale
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7

Rappresentazione di numeri naturali

- Quante cifre sono necessarie per rappresentare in numero X in base 2?
  - Se si usano k bit/cifre, si hanno 2^k possibili configurazioni, da 0…000 a 1…111
  - Si possono pertanto rappresentare i numeri naturali compresi fra *0* e (2^k-1)

Rappresentazione di numeri naturali

- Tutti i numeri x con $2^{k-1} - 1 < x = 2^k - 1$ richiedono k bit per essere rappresentati
- Anche $2^{k-1} < x + 1 = 2^k$
- Quindi per rappresentare il numero x occorrono k bit con

$$k = \lceil \log_2(x+1) \rceil$$

Cioè l'intero immediatamente superiore a

$$\log_2(x+1)$$

Rappresentazione di numeri naturali

- Il risultato vale anche per una generica base b

$$k = \lceil \log_b(x+1) \rceil$$

- Se B è il numero di cifre binarie che serve a rappresentare il numero x e D è il numero di cifre necessarie sempre per rappresentare x

$$B/D = \lceil \log_2(x+1) \rceil / \lceil \log_{10}(x+1) \rceil =$$
$$\lceil \log_{10}(x+1) / \log_{10}(2) \rceil / \lceil \log_{10}(x+1) \rceil \sim 3.3$$

- Occorrono all'incirca 10 cifre binarie per rappresentare 3 cifre decimali.
- Si usano queste abbreviazioni:
  - Kilo (K) = 2^{10} = 1024 ~ 1000
  - Mega (M) = 2^{20} = 1.048.576 ~ 1.000.000
  - Giga (G) = 2^{30} = …… ~ 1.000.000.000
  - Tera (T) = 2^{40} = …… ~ 1.000.000.000.000

Dinamica

- Per numeri interi con *K* cifre binarie
- Dinamica pari *a 2 alla k -1*, $2^{(k-1)}$
- Numero di valori rappresentati pari a 2 alla k

Numeri reali, Rappresentazione in virgola fissa

- Supponendo di utilizzare n=5 bit per la parte intera e m=3 bit per la parte frazionaria:

xxxxx.xxx

Per esempio il numero

11000.101

corrisponde a:

$$1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^{-1} + 1 \cdot 2^{-3} = 24.625$$

Numeri reali, Rappresentazione in virgola fissa

- La rappresentazione in virgola fissa limita fortemente l'intervallo numerico utilizzabile. In tale caso è possibile i rappresentare numeri da 2^{-3}=0.125 a 2^{5}=32
- Il numero di cifre significative dipende dal suo valore assoluto:

  10011.101 = 19.625 ha 8 cifre significative
- Anche

  1001.1101 =9.81125 ha 8 cifre significative

Ma nella rappresentazione in virgola fissa diventa (5:3):

01001.110=9.75

con notevole perdita di precisione.

Precisione, P

- Ea = Vv – Vr Errore assoluto
- Er = Ea / Vv Errore relativo
- Er% = Er 100 Errore relativo percentuale

- Per *m* cifre dopo la virgola
- 2 alla –*m*, 2^{-m} , per la parte frazionaria
- Confronto della precisione *P* con un esempio reale, verifica che la P >= Ea

Operazioni sui numeri binari

- Come per i numeri decimali ma con le seguenti tabelle:

Somma:

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	10

Prodotto:

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Operazioni sui numeri binari

- **Somma:**

0+0=0
0+1=1
1+0=1
1+1=0 con riporto 1 (2_{10})
1+1+1=1 con riporto 1 (3_{10})

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	10

```
111      riporto
 10110+
 11101
------
110011
```

Operazioni sui numeri binari

- **Sottrazione:**

0-0=0
1-0=1
1-1=0
0-1=1
con un prestito dal bit più a sinistra.

A	B	A-B
0	0	0
0	1	1(1)
1	0	1
1	1	0

```
        prestito
  1110-
  0101
  ----
  1001
```

```
        prestito
  1100
  1001
  ----
  0011
```

Operazioni sui numeri binari

- **Prodotto:**

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

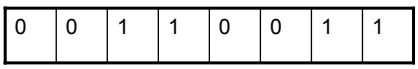
```
     1011
     1101
    ------
     1011+
    0000-
    ------
    01011+
    1011--
    ------
    110111+
    1011---
    -------
    10001111
```

Operazioni sui numeri binari

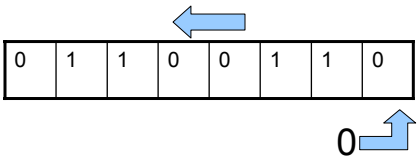
- **Divisione:**

```
101101 | 11
00     | 01111
101
011
 101
  11
 100
  11
  011
   11
   00
```

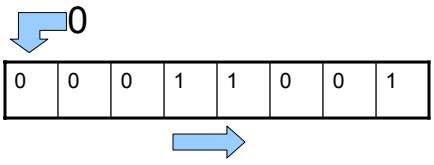
Operazioni di Shift (Scorrimento)



- **Shift a sinistra**: moltiplica per 2



- **Shift a destra**: divide per 2



Complemento

- Complemento a *b* (base):
  - Dato un numero x con base b, e k cifre è definito come

$$C_b(x) = b^k - x$$

- Complemento a *b-1:*

$$C_{b-1}(x) = b^k - 1 - x = C_b(x) - 1$$

Complemento, proprietà

- $C_b(x) = C_{b-1}(x) + 1$
- $C_b\ (C_b\ (x)) = x$
- $C_{b-1}\ (C_{b-1}\ (x)) = x$

$C_2(x) = C_1(x) + 1$

Complemento

- Complemento a 2: $2^k - x$
  - Es: x= 01010110 k=8
    100000000 -
    01010110 =
    010101010
- Complemento a 1: $2^k - 1 - x$
  - Es: x= 01010110 k=8
    11111111 -
    01010110 =
    10101001

Complemento

- Complemento a 1: $2^k - 1 - x$
  - Lo si ottiene semplicemete scambiando 0 con 1 e 1 con 0
  - Es: x= 01010110 k=8
    01010110 x
    10101001 complemento di x (= not x)
- Complemento a 2: $2^k - x$
  - Si fa il complemento ad 1 di x e si somma 1 (complemento veloce)
  - Es: x= 01010110 k=8
    01010110 x
    10101001 + complemento a 1
    1 = sommo 1
    10101010 complemento a 2

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 1c, Rappresentazioni

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2007

Rappresentazione dell'Informazione

In un calcolatore si possono rappresentare vari tipi di informazioni:

- Numeri naturali (visto)
- Numeri reali in virgola fissa (visto)
- Numeri interi (da vedere in questa sessione)
  - In varie forme
- Rappresentazione ASCII
- …
- …

Rappresentazione dei numeri nei calcolatori

- La memoria è organizzata in celle (“parole”) con un numero fisso di bit (ad esempio 8 = 1 byte).

0	1	0	1	1	0	0	1

- Corrisponde a 1 • 64 + 1 • 16 + 1 • 8 +1
- Usando 8 cifre bit si possono rappresentare numeri compresi fra 0 e 2^8-1 = 255

Rappresentazione dei numeri nei calcolatori

L’uso un numero finito di cifre porta ad una aritmetica modulare, che ritorna in forma chiusa oltre la dinamica al numero piu’ basso.

Incrementi unitari.

```
   00000000  (0)
   00000001  (1)
   00000010  (2)
   ……...
   ……...
   11111111  (255)
(1)00000000  (256 -> 0)
   00000001  (1)
   ……...
```

•Quando si raggiunge il numero 255 (tutti 1) non potendo rappresentare il 256 che richiederebbe 9 bit si torna allo 0.

•Si ha quindi una aritmetica modulo 256.

•In generale se si usano parole di k bit si ha una aritmetica modulo *2 alla* ***k***.

Rappresentazione degli interi

- Disponendo di *k* bit si possono avere 2^k configurazioni diverse.
- Metà possono essere usate per i numeri positivi, l'altra metà per quelli negativi.
- Ci sono due possibili rappresentazioni:
  - in modulo e segno
  - in complemento a 2

Rappresentazione degli interi in modulo e segno

- Il bit più significativo viene usato per rappresentare il segno:
  - 0 per i numeri positivi
  - 1 per quelli negativi
- Le cifre restanti rappresentano il modulo

  $+5_{10}$ ->00000101_2

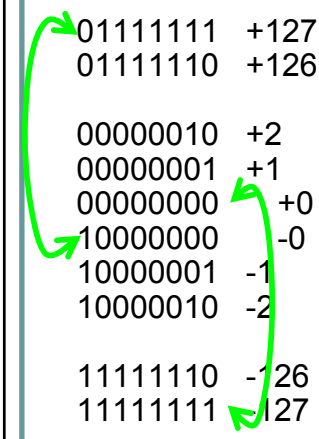
  -10_{10}->10001010_2

  modulo

  segno
- Lo zero ha due rappresentazioni:

  (+0) 00000000 e (-0) 10000000

Rappresentazione degli interi in modulo e segno



- Se si aggiunge 1 a 127 si ha -0
- Se si "aggiunge" 1 a -127 si ha 0 con un traboccamento

Rappresentazione degli interi relativi in modulo e segno

- Algoritmo di **somma**:
  - Confrontare i bit di segno dei due numeri
  - Se sono uguali:
    - Somma i moduli
    - Assegna come bit di segno del risultato il bit di segno degli *operandi*
  - Altrimenti:
    - Confronta i valori assoluti dei due numeri
    - Assegna come bit di segno del risultato quello dell'operando con modulo maggiore
    - Operare la sottrazione fra i moduli nell'ordine giusto
- Macchinoso!! (Analoga la sottrazione).

Rappresentazione dei numeri interi in complemento a 2

- Se hanno numeri di *k* bit
- I numeri positivi sono rappresentati dal loro modulo e hanno il bit più significativo (segno)= 0
- I numeri negativi sono rappresentati facendo il complemento a due del corrispondente numero positivo. Hanno il bit più significativo (segno) = 1
- Si rappresentano come positivi i numeri da 0 a $2^{k-1}-1$
- Si rappresentano come negativi i numero che vanno da -2^{k-1} a -1

Rappresentazione dei numeri interi in complemento a 2

01111111	+127
01111110	+126
00000010	+2
00000001	+1
00000000	0
11111111	-1
11111110	-2
10000001	-127
10000000	-128

Supponendo di avere una cella di 8 bit ($k=8$, $2^k=256$)

$2^{k-1}-1 = 127$

$-2^{k-1} = -128$

Rappresentazione ciclica:

127+1 sconfina a -128 e viceversa

Rappresentazione dei numeri in complemento a 2

- Se si desidera calcolare A-B con numeri di k bit.
- È possibile farlo utilizzando il complemento a 2 di B come numero -B, Infatti:

  $A - B = A - B + 2^K - 2^K = A + 2^K - B - 2^K$

  poiché $C_2(B) = 2^K - B$ si ha:

  $A - B = A + C_2(B) - 2^K$

- Per fare una sottrazione si somma il complemento a 2 del secondo operando
- 2^K è un 1 seguito da k zeri, non rappresentabile con k bit e permette di trascurare i traboccamenti.

Rappresentazione dei numeri in complemento a 2

- interpretare un numero in complemento a due?
- Se il bit più significativo (segno) =0 allora il numero è positivo e le sue cifre ci danno il modulo
  - Es: 00110011 = + 51 (32+16+2+1)
- Se il bit più significativo (segno) =1 allora il numero è negativo e per avere il modulo:

Es: 10110011 negativo

- **01001100+** complemento a 1
- **1**
- **01001101** complemento a 2 = 77 (64+8+4+1)
- il numero originale era -77

Rappresentazione dei numeri in complemento a 2

- Esempi (se k=5, è possibile rappresentare i numeri da -16 a + 15):

```
 01001+   +9
 00100=   +4
 01101   +13
```

```
 01001+   +9 (si noti che si somma per sottrarre)
 11100=   -4
1100101   +5 (traboccamento sul sesto bit eliminato
          grazie al 2^K che in questo caso è 100000)
```


Rappresentazione dei numeri in complemento a 2

- Esempi (se k=5, numeri da -16 a + 15):

```
 10111+   -9
 00100=   +4
 11011   -5
```

```
 10111+   -9
 11100=   -4
1110011   -13 (sesto bit eliminato)
```

Rappresentazione dei numeri in complemento a 2

- Esempi (k=5, numeri da -16 a + 15):

```
01001+   +9
01000=   +8
10001    OVERFLOW + 17 non è
         rappresentabile con 5 bit
         (riporto sul bit di segno)

10111+   -9
10111=   -9
101110   -18 (sesto bit eliminato)
         OVERFLOW -18 non è
         rappresentabile con 5 bit
         (riporto fuori dal bit di segno)
```

Rappresentazione dei numeri in complemento a 2

OVERFLOW (Superamento della dinamica)

- Si ha quando il risultato non è rappresentabile con il numero di bit disponibili.
- Si ha solo sommando due numeri entrambi positivi o entrambi negativi.
- In questo caso il segno del risultato risulta opposto a quello degli operandi.

Esercizi

1) Calcolare A+B e A AND B con A=$(10111011)_2$ e B=$(00110110)_2$
2) Calcolare A-B e A OR B con A=$(10000001)_2$ e B=$(00000101)_2$
3) Calcolare AxB con A=$(10001010)_2$ e B=$(1001)_2$
4) Calcolare A/B con A=$(1100)_2$ e B=$(101)_2$
5) Si converta da decimale ad ottale il numero 426
6) A quale numero ottale corrisponde $(101110110)_2$?
7) Si convertano i seguenti numeri esadecimali in binari e decimali:
   (a) $(AF5)_{16}$ (b) $(FE8)_{16}$
8) Convertire in binario il numero -1238 utilizzando 16 bit per la rappresentazione.

Codici

- Codice: sistema di simboli che permette la rappresentazione dell'informazione
- Esempi: 
- Decodifica agevole vs codici compressi

Definizioni

- **SIMBOLO**: entità di cui non si da qui una definizione formale
- **STRINGA**: sequenza finita di simboli giustapposti (lunghezza della stringa, stringa vuota)
- **ALFABETO**: insieme finito di simboli
- **LINGUAGGIO**: insieme di stringhe di simboli tratti da un alfabeto

Esempi di alfabeti

- Alfabeto italiano:
  {A, B, C, D, …Z}
- Alfabeto greco:
  {α, β, γ, δ, ...ω}
- Alfabeto binario:
  {0, 1}

Alfabeto usato dal calcolatore

- Interruttore (aperto/chiuso)
- Foro su scheda (aperto/chiuso)
- Transistor (in conduzione/spento)
- Tensione (alta/bassa)
- Dominio di magnetizzazione (↑/↓)
- …..

Alfabeto usato dal calcolatore

- Gli elaboratori utilizzano una logica e un'aritmetica **binaria**
- Ai due stati di un dispositivo vengono associati i due simboli **0** e **1**

Codice ASCII

- ASCII: American Standard Code for Information Interchange
- 7 bit quindi 128 simboli diversi
- ASCII esteso (8bit)
  - diverse estensioni in dipendenza dal paese
  - oppure aggiunge la parità

Tabella Ascii

ASCII Codes

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01	☺	SOH
^B	2	02	☻	STX
^C	3	03	♥	ETX
^D	4	04	♦	EOT
^E	5	05	♣	ENQ
^F	6	06	♠	ACK
^G	7	07	•	BEL
^H	8	08	◘	BS
^I	9	09	○	HT
^J	10	0A	◙	LF
^K	11	0B	♂	VT
^L	12	0C	♀	FF
^M	13	0D	♪	CR
^N	14	0E	♫	SO
^O	15	0F	☼	SI
^P	16	10	►	DLE
^Q	17	11	◄	DC1
^R	18	12	↕	DC2
^S	19	13	‼	DC3
^T	20	14	¶	DC4
^U	21	15	§	NAK
^V	22	16	▬	SYN
^W	23	17	↨	ETB
^X	24	18	↑	CAN
^Y	25	19	↓	EM
^Z	26	1A	→	SUB
^[	27	1B	←	ESC
^\	28	1C	∟	FS
^]	29	1D	↔	GS
^^	30	1E	▲	RS
^_	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	Δ
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	▒
177	B1	▒
178	B2	▒
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	▒
220	DC	▒
221	DD	▒
222	DE	▒
223	DF	▒
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	ϕ
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	ⁿ
253	FD	²
254	FE	■
255	FF	

Tabella Ascii

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

Paolo Nesi, Univ. Firenze, Italy, 2003-07 134

Codice ASCII

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com

Codice ASCII - Esteso

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	│	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

Source: www.asciitable.com

Conversioni con Esadecimali

Hexadecimal-Binary-Decimal Conversion

Hex Number	Binary Number	Decimal Digit 000X	Decimal Digit 00X0	Decimal Digit 0X00	Decimal Digit X000
0	0000	0	0	0	0
1	0001	1	16	256	4,096
2	0010	2	32	512	8,192
3	0011	3	48	768	12,288
4	0100	4	64	1,024	16,384
5	0101	5	80	1,280	20,480
6	0110	6	96	1,536	24,576
7	0111	7	112	1,792	28,672
8	1000	8	128	2,048	32,768
9	1001	9	144	2,304	36,864
A	1010	10	160	2,560	40,960
B	1011	11	176	2,816	45,056
C	1100	12	192	3,072	49,152
D	1101	13	208	3,328	53,248
E	1110	14	224	3,584	57,344
F	1111	15	240	3,840	61,440

Unicode Character Code

- Unicode is a 16-bit code.

0000	NUL	0020	SP	0040	@	0060	`	0080	Ctrl	00A0	NBS	00C0	À	00E0	à
0001	SOH	0021	!	0041	A	0061	a	0081	Ctrl	00A1	¡	00C1	Á	00E1	á
0002	STX	0022	"	0042	B	0062	b	0082	Ctrl	00A2	¢	00C2	Â	00E2	â
0003	ETX	0023	#	0043	C	0063	c	0083	Ctrl	00A3	£	00C3	Ã	00E3	ã
0004	EOT	0024	$	0044	D	0064	d	0084	Ctrl	00A4	¤	00C4	Ä	00E4	ä
0005	ENQ	0025	%	0045	E	0065	e	0085	Ctrl	00A5	¥	00C5	Å	00E5	å
0006	ACK	0026	&	0046	F	0066	f	0086	Ctrl	00A6	¦	00C6	Æ	00E6	æ
0007	BEL	0027	'	0047	G	0067	g	0087	Ctrl	00A7	§	00C7	Ç	00E7	ç
0008	BS	0028	(	0048	H	0068	h	0088	Ctrl	00A8	¨	00C8	È	00E8	è
0009	HT	0029	)	0049	I	0069	i	0089	Ctrl	00A9	©	00C9	É	00E9	é
000A	LF	002A	*	004A	J	006A	j	008A	Ctrl	00AA	ª	00CA	Ê	00EA	ê
000B	VT	002B	+	004B	K	006B	k	008B	Ctrl	00AB	«	00CB	Ë	00EB	ë
000C	FF	002C	´	004C	L	006C	l	008C	Ctrl	00AC	¬	00CC	Ì	00EC	ì
000D	CR	002D	-	004D	M	006D	m	008D	Ctrl	00AD	-	00CD	Í	00ED	í
000E	SO	002E	.	004E	N	006E	n	008E	Ctrl	00AE	®	00CE	Î	00EE	î
000F	SI	002F	/	004F	O	006F	o	008F	Ctrl	00AF	¯	00CF	Ï	00EF	ï
0010	DLE	0030	0	0050	P	0070	p	0090	Ctrl	00B0	°	00D0	Ð	00F0	ð
0011	DC1	0031	1	0051	Q	0071	q	0091	Ctrl	00B1	±	00D1	Ñ	00F1	ñ
0012	DC2	0032	2	0052	R	0072	r	0092	Ctrl	00B2	²	00D2	Ò	00F2	ò
0013	DC3	0033	3	0053	S	0073	s	0093	Ctrl	00B3	³	00D3	Ó	00F3	ó
0014	DC4	0034	4	0054	T	0074	t	0094	Ctrl	00B4	´	00D4	Ô	00F4	ô
0015	NAK	0035	5	0055	U	0075	u	0095	Ctrl	00B5	µ	00D5	Õ	00F5	õ
0016	SYN	0036	6	0056	V	0076	v	0096	Ctrl	00B6	¶	00D6	Ö	00F6	ö
0017	ETB	0037	7	0057	W	0077	w	0097	Ctrl	00B7	·	00D7	×	00F7	÷
0018	CAN	0038	8	0058	X	0078	x	0098	Ctrl	00B8	¸	00D8	Ø	00F8	ø
0019	EM	0039	9	0059	Y	0079	y	0099	Ctrl	00B9	¹	00D9	Ù	00F9	ù
001A	SUB	003A	:	005A	Z	007A	z	009A	Ctrl	00BA	º	00DA	Ú	00FA	ú
001B	ESC	003B	;	005B	[	007B	{	009B	Ctrl	00BB	»	00DB	Û	00FB	û
001C	FS	003C	<	005C	\	007C	|	009C	Ctrl	00BC	1/4	00DC	Ü	00FC	ü
001D	GS	003D	=	005D	]	007D	}	009D	Ctrl	00BD	1/2	00DD	Ý	00FD	Þ
001E	RS	003E	>	005E	^	007E	~	009E	Ctrl	00BE	3/4	00DE	ý	00FE	þ
001F	US	003F	?	005F	_	007F	DEL	009F	Ctrl	00BF	¿	00DF	ß	00FF	ÿ

NUL	Null	SOH	Start of heading	CAN	Cancel	SP	Space
STX	Start of text	EOT	End of transmission	EM	End of medium	DEL	Delete
ETX	End of text	DC1	Device control 1	SUB	Substitute	Ctrl	Control
ENQ	Enquiry	DC2	Device control 2	ESC	Escape	FF	Form feed
ACK	Acknowledge	DC3	Device control 3	FS	File separator	CR	Carriage return
BEL	Bell	DC4	Device control 4	GS	Group separator	SO	Shift out
BS	Backspace	NAK	Negative acknowledge	RS	Record separator	SI	Shift in
HT	Horizontal tab	NBS	Non-breaking space	US	Unit separator	DLE	Data link escape
LF	Line feed	ETB	End of transmission block	SYN	Synchronous idle	VT	Vertical tab

Prefissi Scientifici

- For computer memory, 1K = 2^{10} = 1024. For everything else, like clock speeds, 1K = 1000, and likewise for 1M, 1G, *etc.*

Prefix	Abbrev.	Quantity	Prefix	Abbrev.	Quantity
milli	m	10^{-3}	Kilo	K	10^{3}
micro	μ	10^{-6}	Mega	M	10^{6}
nano	n	10^{-9}	Giga	G	10^{9}
pico	p	10^{-12}	Tera	T	10^{12}
femto	f	10^{-15}	Peta	P	10^{15}
atto	a	10^{-18}	Exa	E	10^{18}

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

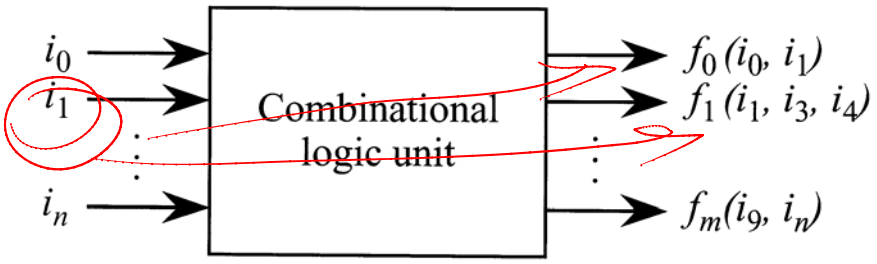
Parte 2, Logica Combinatoria

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2007

Logica Combinatoria

- Un circuito logico combinatorio produce il valore delle sue uscite solo in base al valore degli ingressi in quel "momento"
- La stima dell'uscita non tiene conto dell' evoluzione temporale degli ingressi
- "Momento" significa che in circuiti reali si deve dare il tempo al circuito di raggiungere un valore stabile

Logica Combinatoria



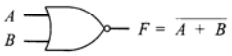
- In seguito vedremo anche la logica sequenziale

Altre Porte oltre a: and, or, not

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

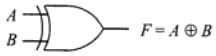
NAND

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



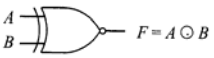
NOR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



Exclusive-OR (XOR)

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

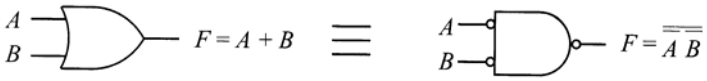


Exclusive-NOR (XNOR)

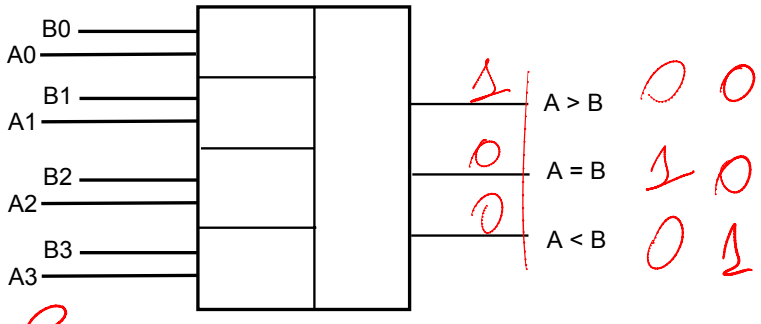
Teorema di De Morgan

A B	$\overline{A\,B} = \overline{A} + \overline{B}$		$\overline{A + B} = \overline{A}\,\overline{B}$	
0 0	1	1	1	1
0 1	1	1	0	0
1 0	1	1	0	0
1 1	0	0	0	0

$$A + B = \overline{\overline{A + B}} = \overline{\overline{A}\,\overline{B}}$$



Comparatore Modulare



A, B

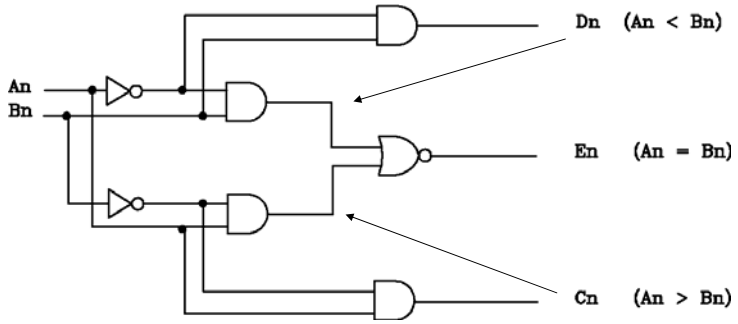
Comparatore a 4 bit

- A3-A2-A1-A0 che si confronta con B3-B2-B1-B0
- A=B
  - A3=B3, A2=B2, A1=B1, A0=B0
  - Le condizioni che devono essere vere insieme sono 4
  - E=E3 E2 E1 E0
- A>B
  - Caso0: A3>B3 => A3 not B3
  - Caso1: A3=B3, A2>B2 => E3 A2 not B2
  - Caso2: A3=B3, A2=B2, A1>B1 => E3 E2 A1 not B1
  - Caso3: A3=B3, A2=B3, A1=B1, A0>B0 => E3 E2 E1 A0 not B0
  - A e' maggiore di B = Caso0 or Caso1 or Caso2 or Caso3
- A<B
  - ……
  - ….
  - …
- Molto complessa come soluzione, e' meglio studiare una soluzione modulare….

Paolo Nesi, Univ. Firenze, Italy, 2003-07 147

Comparatore digitale a due bit

- Realizzazione di un comparatore a due ingressi e 3 uscite
  - En vera se An = Bn
  - Dn vera se An < Bn
  - Cn vera se An > Bn



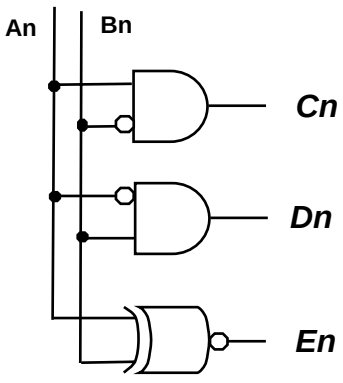
Paolo Nesi, Univ. Firenze, Italy, 2003-07 148

Comparatore Digitale

- Dn = not A B vero se An<Bn
- Cn = not B A vero se Bn>An
- En = A B + not A not B = not (A xor B)

		<		>
An	***Bn***	***Dn***	***En***	***Cn***
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Singolo Modulo Comparatore



- Dn = not A B
- Cn = not B A
- En = A B + not A not B = not (A xor B)

Se fosse a 2 bit !

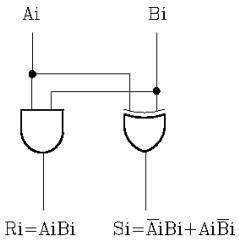
>	>	<	<	>	<
C1	C0	D1	D0	C	D
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0

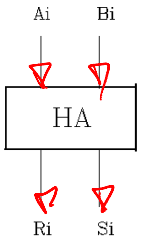
- C = not C1 C0 not D1 not D0 +
  C1 not C0 not D1 not D0 +
  C1 not C0 not D1 D0 +
  C1 C0 not D1 not D0 +
  C1 C0 not D1 D0 +
  C1 C0 D1 not D0 =
- = C1 not D1 + C0 not D1 not D0 + C1 C0 not D0

Molti termini da portare avanti, non conviene seguire questa strada, specialmente se la necessita' e' quella di produrre un comparatore a N bit

Sommatore, Half Adder

Ai	Bi	Si	Ri
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

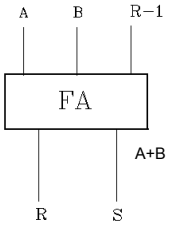


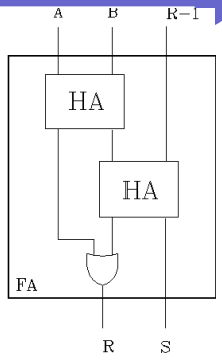


- Verifica della scrittura delle equazioni per Si e Ri
- Si = somma, Sum
- Ri = riporto, Rest
- Non tiene conto del resto del bit precedente

Full Adder, Sommatore Completo

Ai	Bi	R_{-1}	Si	Ri
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



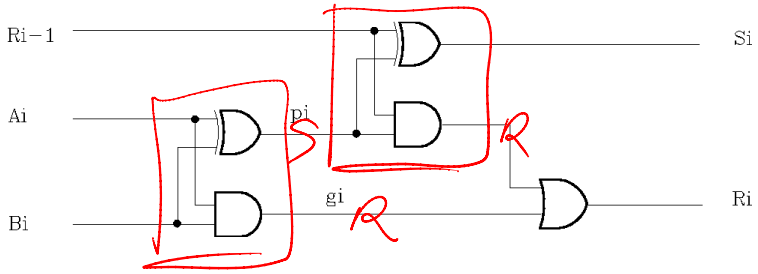


- Si = not A not B R + not A B not R+ A not B not R+ A B R
- Ri = not A B R + A not B R + A B not R + A B R
- Ri = R(not AB+ not BA) + AB

Full Adder, Sommatore Completo

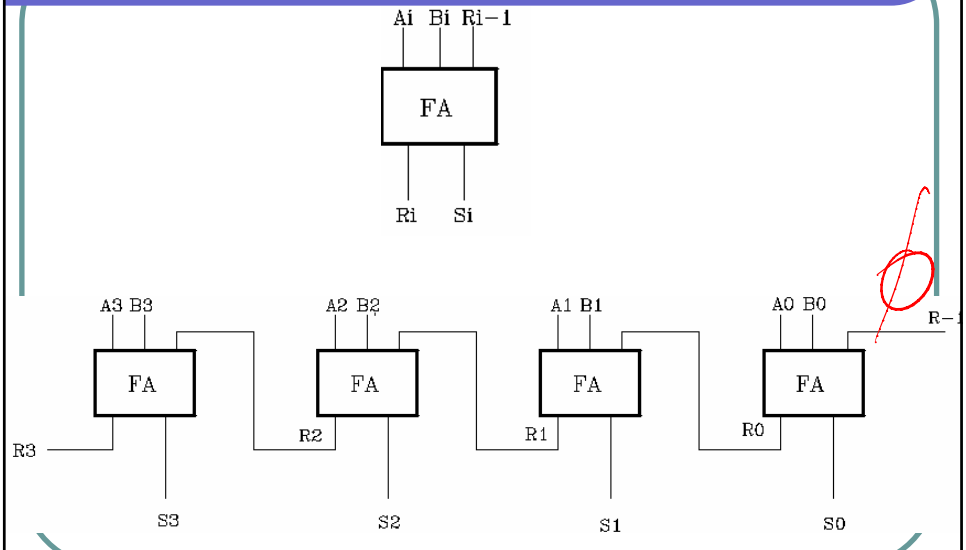
- S = not A not B R + not A B not R+ A not B not R+ A B R
  - Poiche': D xor F = not DF + D not F
  - Poiche': not (D xor F) = not DF + D not F
- Si = not R (A xor B) + R (not A not B+ AB)
- Si = not R (A xor B) + R not (A xor B)
- Ri = R(not AB+ not BA) + AB

Full Adder, Sommatore Completo

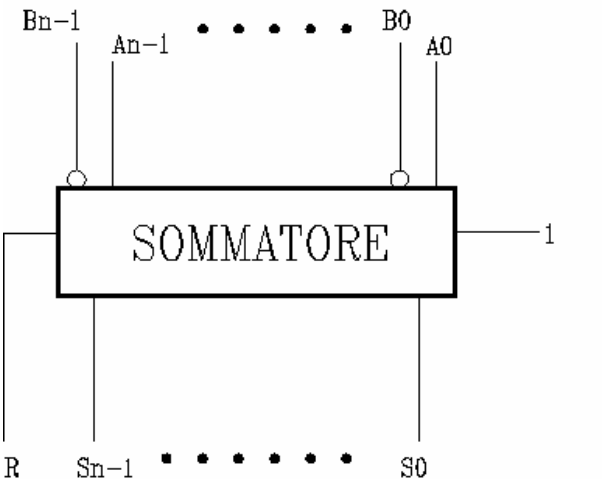


- Si = not R (A xor B) + R not (A xor B)
- Ri = R (A xor B) + AB
- Pi =
- Gi =
- Estrarre equazioni ?

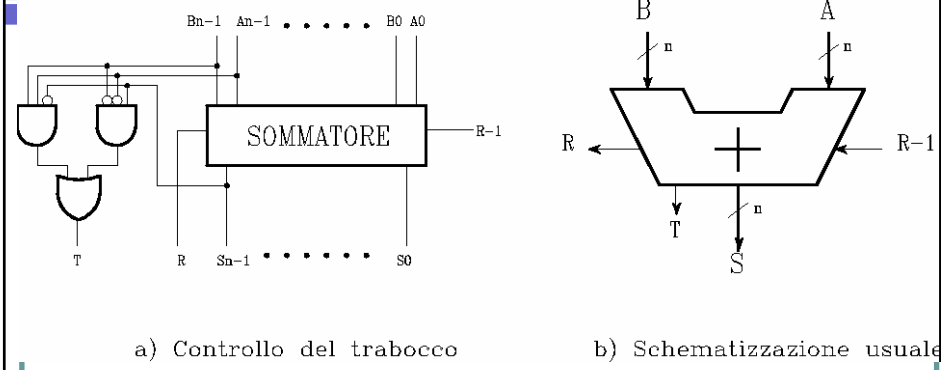
Sommatore Binario a 4 bit



Sommatore



Sommatore controllato

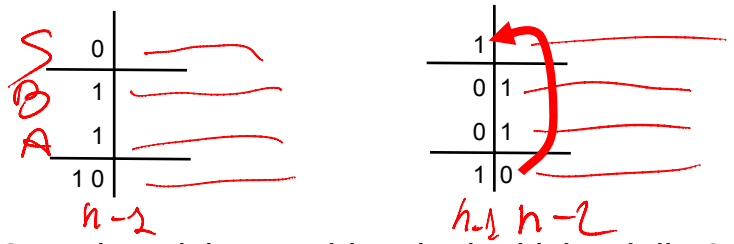


a) Controllo del trabocco b) Schematizzazione usuale

- Controllo del traboccamento, due termini, due casi
- $T = (B_{n-1} A_{n-1} not\ S_{n-1})$ or $(not\ B_{n-1}\ not\ A_{n-1} S_{n-1})$
- Riporto sul bit di segno ma non fuori, oppure
- Riporto fuori dal bit di segno ma non sul segno

Traboccamento, T

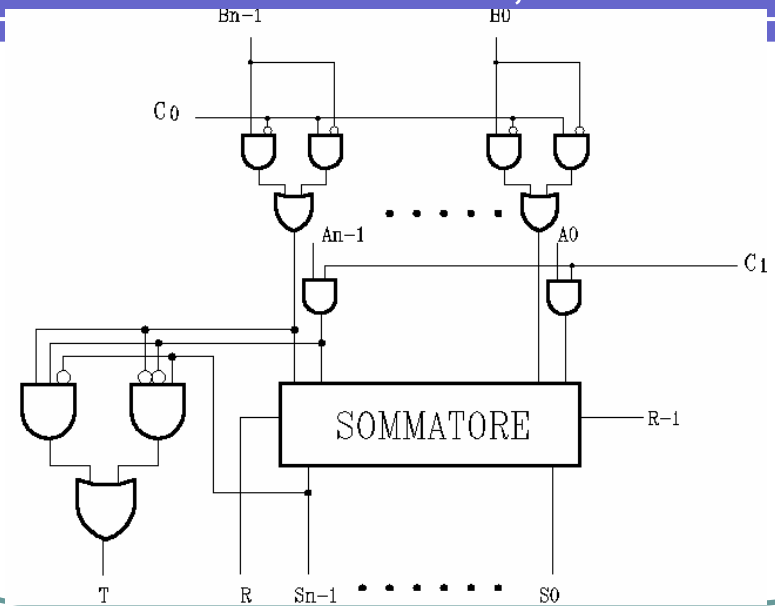
- $T = (B_{n-1}A_{n-1}not\ S_{n-1})$ or $(not\ B_{n-1}\ not\ A_{n-1}S_{n-1})$
- Riporto sul bit di segno ma non fuori, oppure
- Riporto fuori dal bit di segno ma non sul segno



- *Sono le uniche combinazioni critiche delle 8 che vengono fuori dai bit* $B_{n-1}A_{n-1}S_{n-1}$

Paolo Nesi, Univ. Firenze, Italy, 2003-07 161

Sommatore controllato, ALU

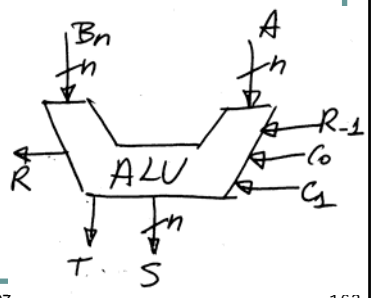


Paolo Nesi, Univ. Firenze, Italy, 2003-07 162

La ALU, controlli e risultati

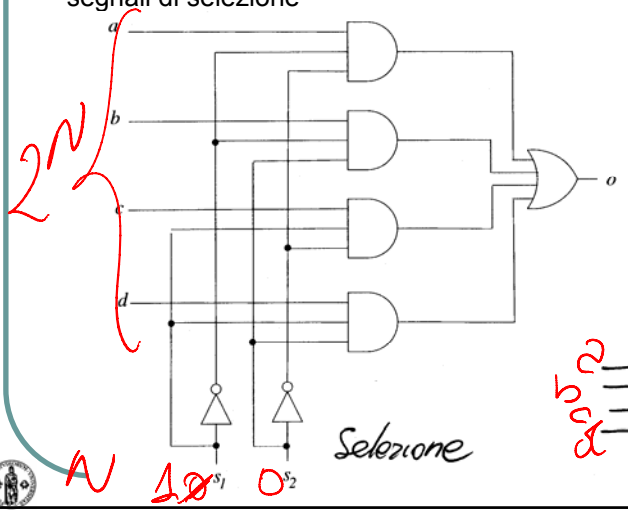
C_2	C_0	R_1	S
0	0	0	B
0	0	1	B+1 incremento
0	1	0	$\overline{B}$ compl. a 1
0	1	1	$\overline{B}+1 = -B$ CHS $C_2(B)$
1	0	0	A+B
1	0	1	A+B+1
1	1	0	$A+\overline{B} = A-B-1$
1	1	1	$A+\overline{B}+1 = A-B$ differenza

$C_2(B)$

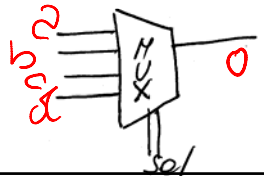


Multiplexer o Selettore

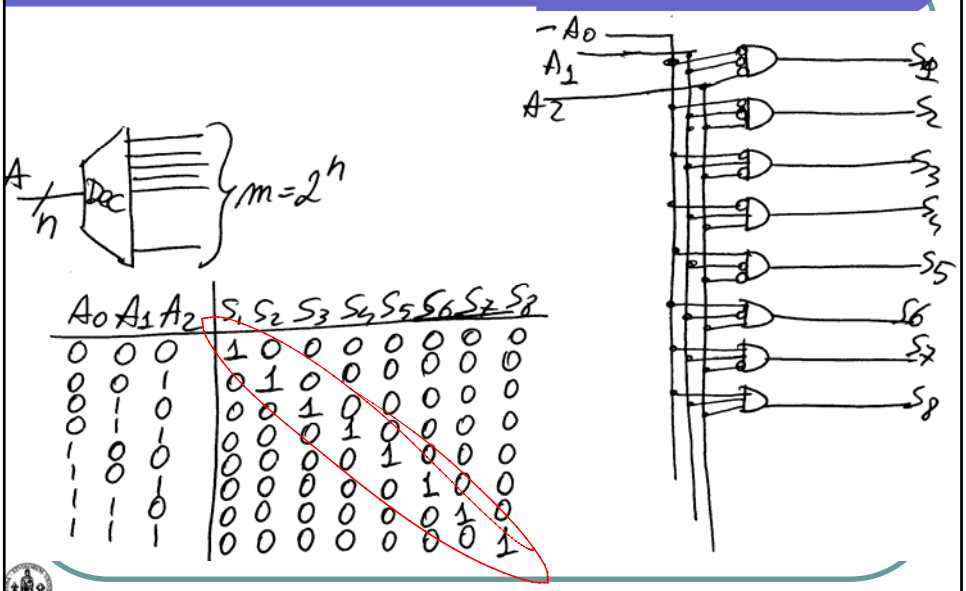
- Lo scopo e' selezionare uno degli ingressi e riportarlo in uscita.
- 2^n segnali di ingresso vengono selezionati in base al valore degli N segnali di selezione



s_1	s_2	o
0	0	a
0	1	b
1	0	c
1	1	d

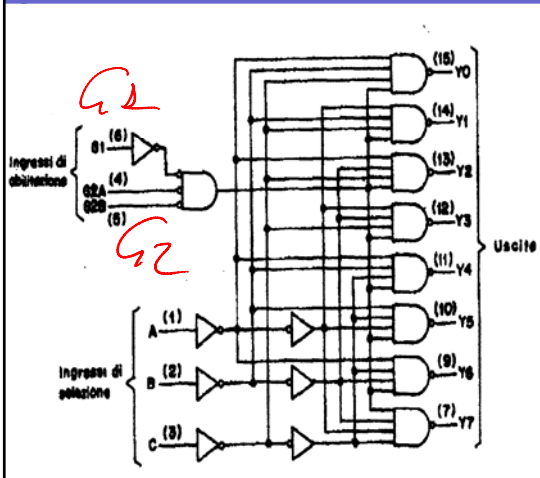


Decoder or Demultiplexer



A0	A1	A2	S0	S1	S2	S3	S4	S5	S6	S7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoder or Demultiplexer LS138



'LS138, 'S138
tabella funzionale

Ingressi					Uscite							
abilitazione		selezione										
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

· Decodificatore/Demultiplexer LS138. Configurazione logica e tabella funzionale.

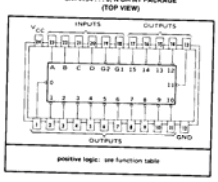
TTL
MSI

**TYPES SN54154, SN74154,
4-LINE-TO-16-LINE DECODERS/DEMULTIPLEXERS**

BULLETIN NO. DL-S 7211806, DECEMBER 1972

- '154 is Ideal for High-Performance Memory Decoding
- Decodes 4 Binary-Coded Inputs into One of 16 Mutually Exclusive Outputs
- Performs the Demultiplexing Function by Distributing Data From One Input Line to Any One of 16 Outputs
- Input Clamping Diodes Simplify System Design
- High Fan-Out, Low-Impedance, Totem-Pole Outputs
- Fully Compatible with Most TTL, DTL, and MSI Circuits

SN54154 . . . J OR W PACKAGE
SN74154 . . . J, N OR NT PACKAGE
(TOP VIEW)



TYPE	TYPICAL AVERAGE PROPAGATION DELAY 3 LEVELS OF LOGIC	STROBE	TYPICAL POWER DISSIPATION
'154	23 ns	19 ns	170 mW

description

Each of these monolithic, 4-line-to-16-line decoders utilizes TTL circuitry to decode four binary-coded inputs into one of sixteen mutually exclusive outputs when both the strobe inputs, G1 and G2, are low. The demultiplexing function is performed by using the 4 input lines to address the output line, passing data from one of the strobe inputs with the other strobe input low. When either strobe input is high, all outputs are high. These demultiplexers are ideally suited for implementing high-performance memory decoders. For ultra-high-speed systems, SN54S138/SN74S138 and SN54S139/SN74S139 are recommended.

These circuits are fully compatible for use with most other TTL and DTL circuits. All inputs are buffered and input clamping diodes are provided to minimize transmission-line effects and thereby simplify system design.

Series 54 devices are characterized for operation over the full military temperature range of −55°C to 125°C; Series 74 devices are characterized for operation from 0°C to 70°C.

TYPES SN54154, SN7415
4-LINE-TO-16-LINE DECODERS/ DEMULTIPLEXER

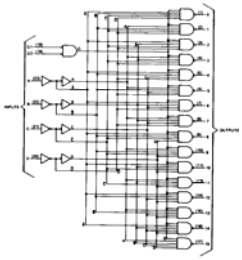
logic

FUNCTION TABLE

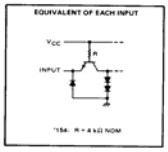
INPUTS					OUTPUTS													
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

H = high level, L = low level, X = irrelevant

functional block diagram and schematics of inputs and outputs

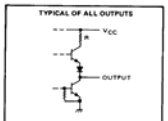


EQUIVALENT OF EACH INPUT



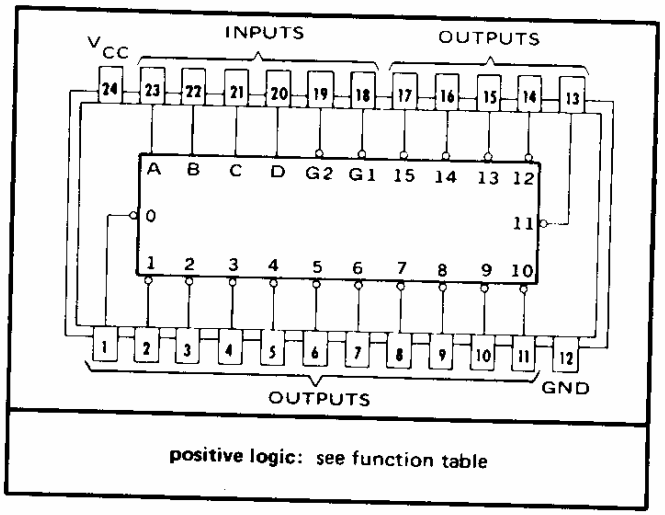
'154: R = 4 kΩ NOM

TYPICAL OF ALL OUTPUTS



'154: R = 130 Ω NOM

SN54154 . . . J OR W PACKAGE
SN74154 . . . J, N OR NT PACKAGE
(TOP VIEW)



logic

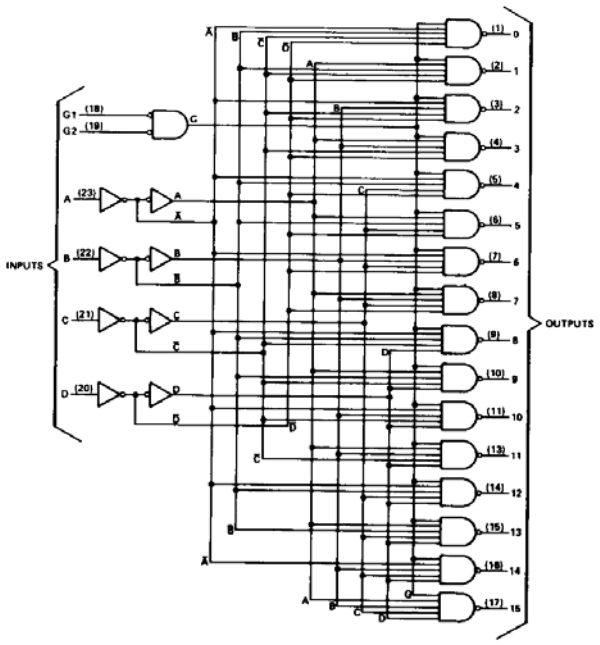
FUNCTION TABLE

INPUTS						OUTPUTS													
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

H = high level, L = low level, X = irrelevant

Paolo Nesi, Univ. Firenze, Italy, 2003-07 169

functional block diagram and schematics of inputs and outputs



170

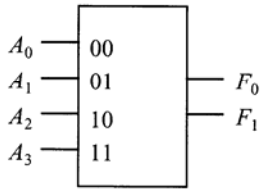
Simulatore di circuiti Logici

- Sulla pagina WEB potete scaricare un simulatore di reti logiche,
- Usatelo per fare esercizio sulle reti combinatorie che abbiamo visto:
  - Realizzare delle reti
  - Presentare delle sequenze di dati in ingresso
  - Verificare la sequenza di uscita in base alle equazioni algebriche e alla tabella della verita'
  - Etc..
  - Etc.

Encoder

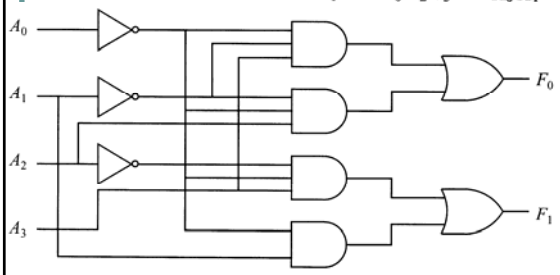
Questo è un Encoder che realizza una codifica a caso.

E' possibile realizzare una numero elevato di diversi encoder, con diverse codifiche e numero di uscite !!!



$$F_0 = \overline{A_0}\,\overline{A_1}\,A_3 + \overline{A_0}\,A_1\,A_2$$

$$F_1 = \overline{A_0}\,A_2\,A_3 + \overline{A_0}\,A_1$$



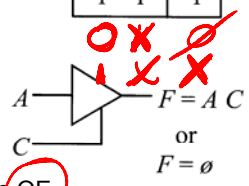
A_0	A_1	A_2	A_3	F_0	F_1
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

NOTA: sul libro trovate solo il concetto di priority Encoder: un tipo particolare di Encoder.

Buffer tri-state

- Ø = Terzo stato, alta impedenza, isolato

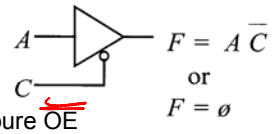
C	A	F
0	0	ø
0	1	ø
1	0	0
1	1	1



C oppure OE

Tri-state buffer

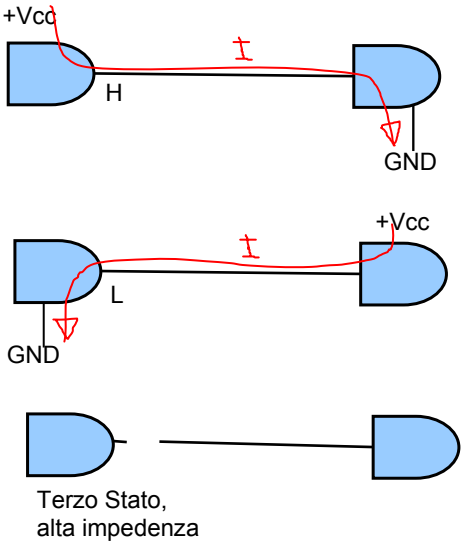
C	A	F
0	0	0
0	1	1
1	0	ø
1	1	ø



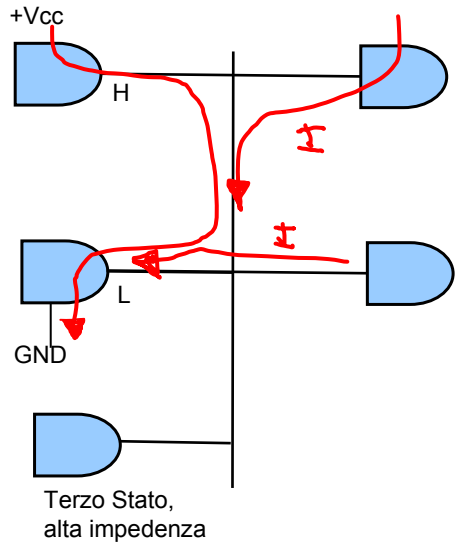
C oppure OE

Tri-state buffer, inverted control

Varie condizioni

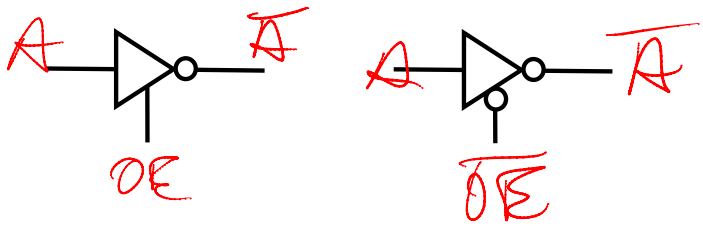


Varie condizioni



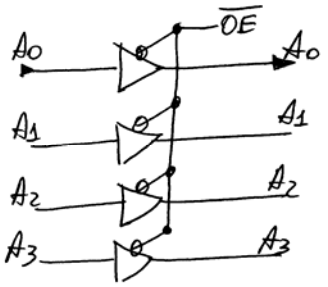
Buffer tri-state negati

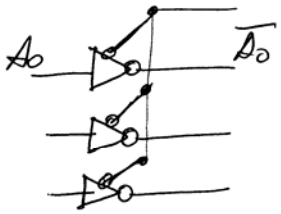
- Ø = Terzo stato, alta impedenza, isolato



Buffer tristate per bus e gruppi di bit

- Puo' essere negato o meno e ha un Output Enable, OE che puo' essere negato o meno
- Si veda lo schema del 240 e del 241





Paolo Nesi, Univ. Firenze, Italy, 2003-07 177

TYPES SN54LS240,SN54LS241,SN54LS244,SN54S240,SN54S241,
SN74LS240,SN74LS241,SN74LS244,SN74S240,SN74S241
OCTAL BUFFERS AND LINE DRIVERS WITH 3-STATE OUTPUTS

- 3-State Outputs Drive Bus Lines or Buffer Memory Address Registers
- P-N-P Inputs Reduce D-C Loading
- Hysteresis at Inputs Improves Noise Margins

description

These octal buffers and line drivers are designed specifically to improve both the performance and density of three-state memory address drivers, clock drivers, and bus-oriented receivers and transmitters. The designer has a choice of selected combinations of inverting and noninverting outputs, symmetrical G (active low output control) inputs, and complementary G and G inputs. These devices feature high fan-out, improved fan-in, and 400-mV noise-margin. The SN74LS' and SN74S' can be used to drive terminated lines down to 133 ohms.

schematics of inputs and outputs

'LS240, 'LS241, 'LS244
EQUIVALENT OF EACH INPUT

'S240 'S241
EQUIVALENT OF EACH INPUT

TYPICAL OF ALL OUTPUTS

'LS240, 'LS241, 'LS244: R = 90 Ω NOM
'S240, 'S241: R = 25 Ω NOM

SN54LS241, SN54S241 . . . J
SN74LS241, SN74S241 . . . J OR N
(TOP VIEW)

SN54LS244 . . . J
SN74LS244 . . . J OR N
(TOP VIEW)

SN54LS240, SN54S240 . . . J
SN74LS240, SN74S240 . . . J OR N
(TOP VIEW)

TEXAS INSTRUMENTS

7-351

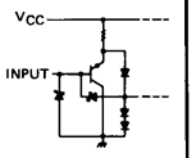
Paolo Nesi, Univ. Firenze, Italy, 2003-07 178

mentary G and $\overline{G}$ inputs. These devices feature high fan-out, improved fan-in, and 400-mV noise-margin. The SN74LS' and SN74S' can be used to drive terminated lines down to 133 ohms.

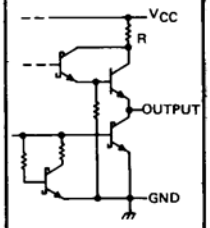
schematics of inputs and outputs

'LS240, 'LS241, 'LS244

EQUIVALENT OF EACH INPUT

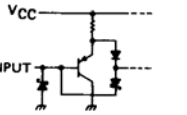


TYPICAL OF ALL OUTPUTS



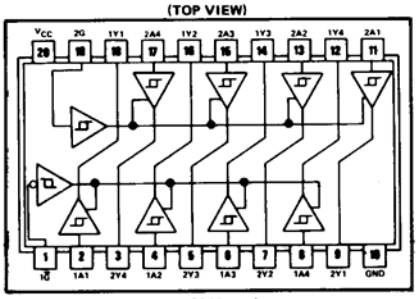
'S240 'S241

EQUIVALENT OF EACH INPUT

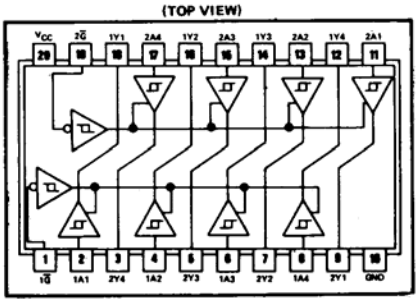


'LS240, 'LS241, 'LS244;
R = 50 Ω NOM
'S240, 'S241:
R = 25 Ω NOM

SN54LS241, SN54S241 . . . J
SN74LS241, SN74S241 . . . J OR N
(TOP VIEW)



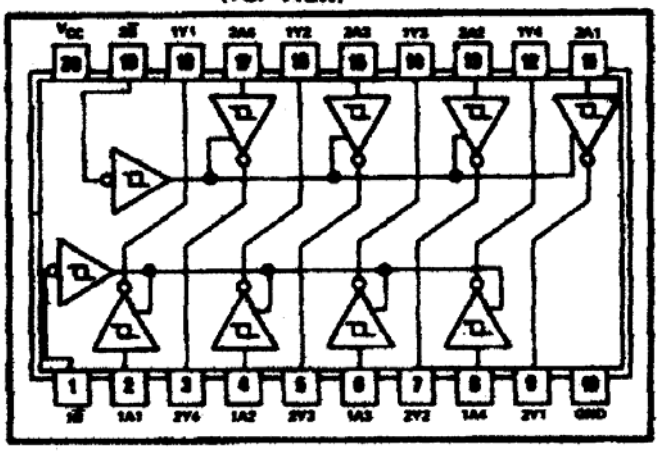
SN54LS244 . . . J
SN74LS244 . . . J OR N
(TOP VIEW)



Paolo Nesi, Univ. Firenze, Italy, 2003-07 179

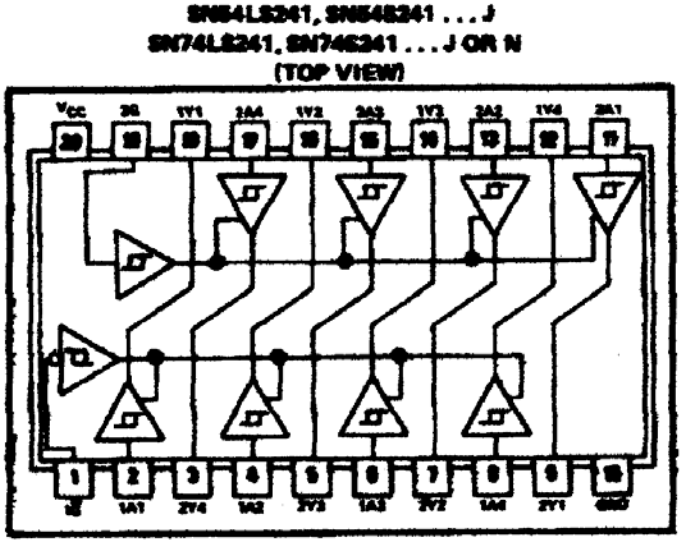
SN54LS240

SN54LS240, SN54S240 . . . J
SN74LS240, SN74S240 . . . J OR N
(TOP VIEW)

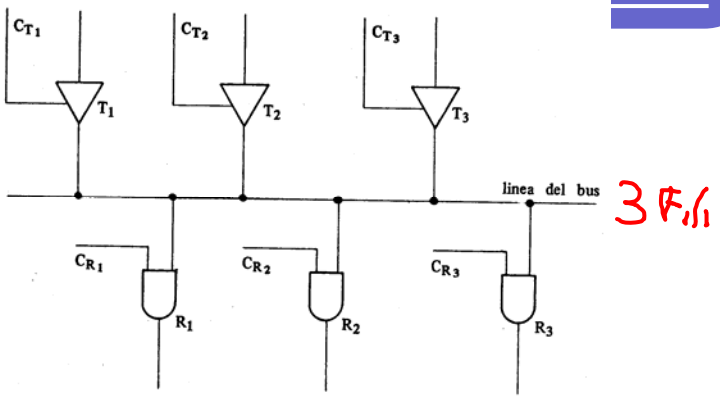


Paolo Nesi, Univ. Firenze, Italy, 2003-07 180

SN54LS241

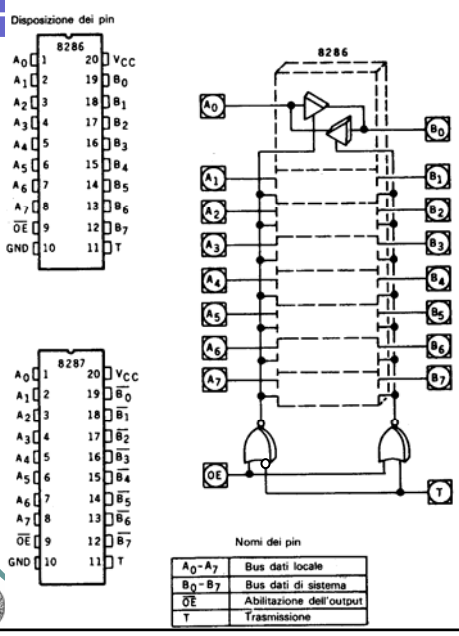


IL BUS con Tri State



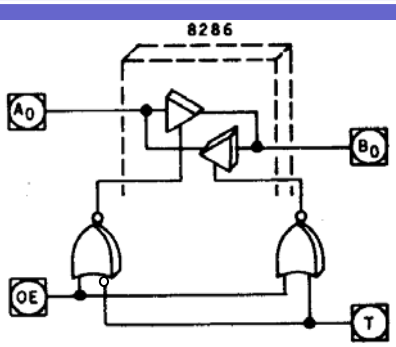
- 3 Buffer tri-state come ingressi al bus che devono essere attivi solo uno alla volta
- 3 porte AND che possono leggere contemporaneamente la stessa linea di BUS

Ricetrasmissione dati, Transceiver



184

Ricetrasmissione dati, Transceiver



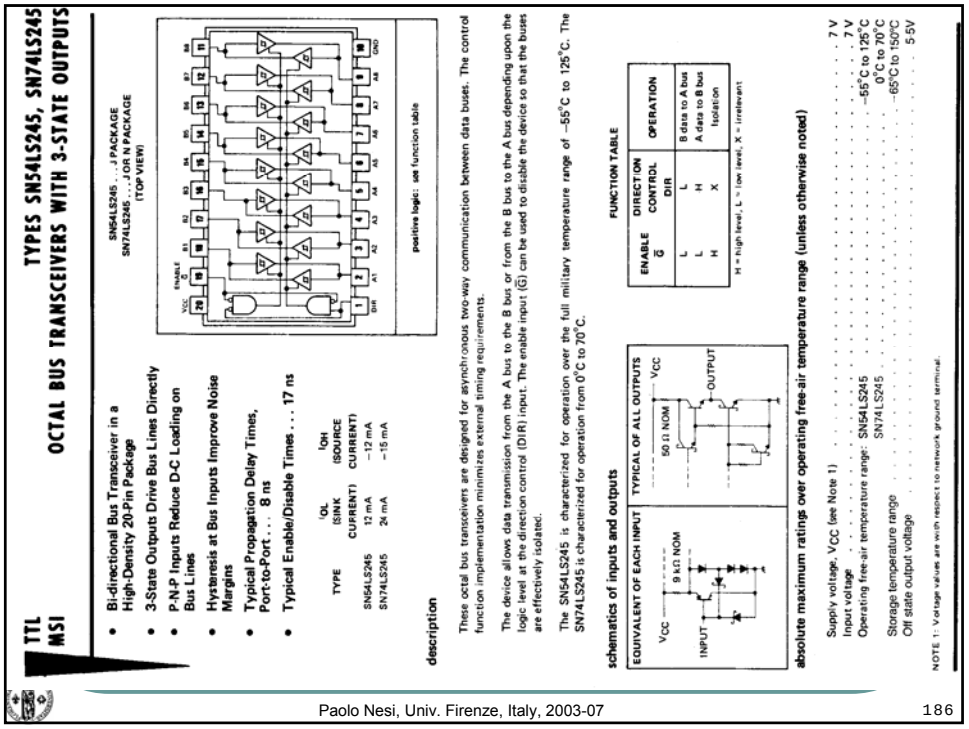
Nomi dei pin

A_0-A_7	Bus dati locale
B_0-B_7	Bus dati di sistema
$\overline{OE}$	Abilitazione dell'output
T	Trasmissione

OE'	T	Operation
0	1	A to B
0	0	B to A
1	X	Alta impedenza

Paolo Nesi, Univ. Firenze, Italy, 2003-07

185



TTL MSI

TYPES SN54LS245, SN74LS245
OCTAL BUS TRANSCEIVERS WITH 3-STATE OUTPUTS

- Bi-directional Bus Transceiver in a High-Density 20-Pin Package
- 3-State Outputs Drive Bus Lines Directly
- P-N-P Inputs Reduce D-C Loading on Bus Lines
- Hysteresis at Bus Inputs Improve Noise Margins
- Typical Propagation Delay Times, Port-to-Port . . . 8 ns
- Typical Enable/Disable Times . . . 17 ns

TYPE	I_{OL} (SINK CURRENT)	I_{OH} (SOURCE CURRENT)
SN54LS245	12 mA	−12 mA
SN74LS245	24 mA	−15 mA

SN54LS245 . . . J PACKAGE
SN74LS245 . . . J OR N PACKAGE
(TOP VIEW)

positive logic: see function table

description

These octal bus transceivers are designed for asynchronous two-way communication between data buses. The control function implementation minimizes external timing requirements.

The device allows data transmission from the A bus to the B bus or from the B bus to the A bus depending upon the logic level at the direction control (DIR) input. The enable input ($\overline{G}$) can be used to disable the device so that the buses are effectively isolated.

The SN54LS245 is characterized for operation over the full military temperature range of −55°C to 125°C. The SN74LS245 is characterized for operation from 0°C to 70°C.

schematics of inputs and outputs

EQUIVALENT OF EACH INPUT — TYPICAL OF ALL OUTPUTS

FUNCTION TABLE

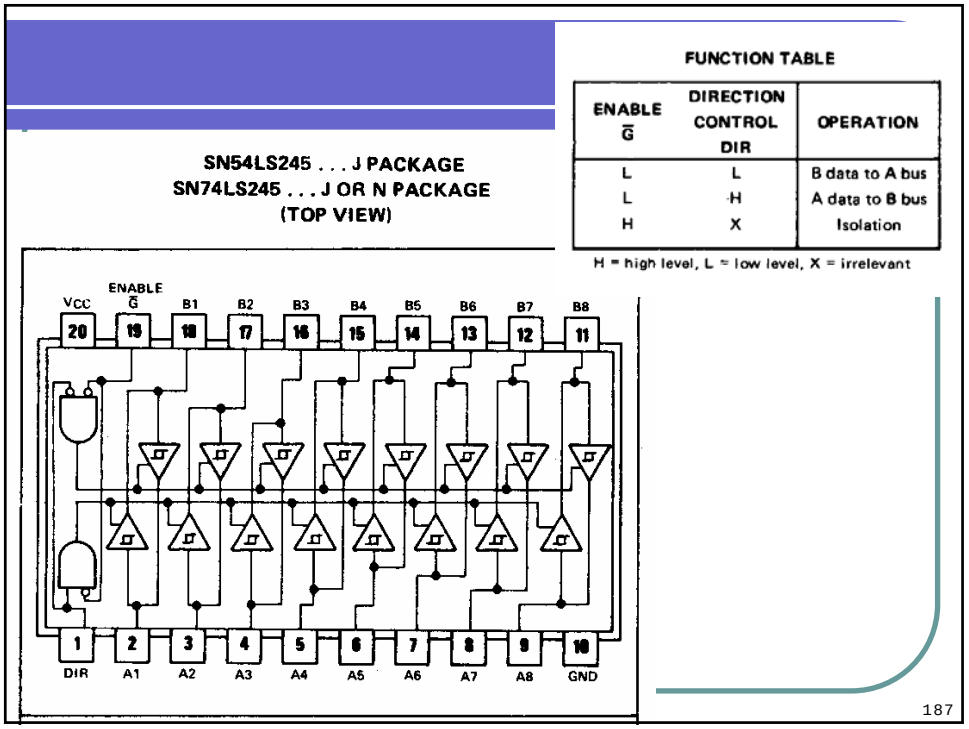
ENABLE $\overline{G}$	DIRECTION CONTROL DIR	OPERATION
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

H = high level, L = low level, X = irrelevant

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . 7 V
Input voltage . . . 7 V
Operating free-air temperature range: SN54LS245 . . . −55°C to 125°C
SN74LS245 . . . 0°C to 70°C
Storage temperature range . . . −65°C to 150°C
Off state output voltage . . . 5.5V

NOTE 1: Voltage values are with respect to network ground terminal.



SN54LS245 . . . J PACKAGE
SN74LS245 . . . J OR N PACKAGE
(TOP VIEW)

FUNCTION TABLE

ENABLE $\overline{G}$	DIRECTION CONTROL DIR	OPERATION
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

H = high level, L = low level, X = irrelevant

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 3, Logica Sequenziale

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2007

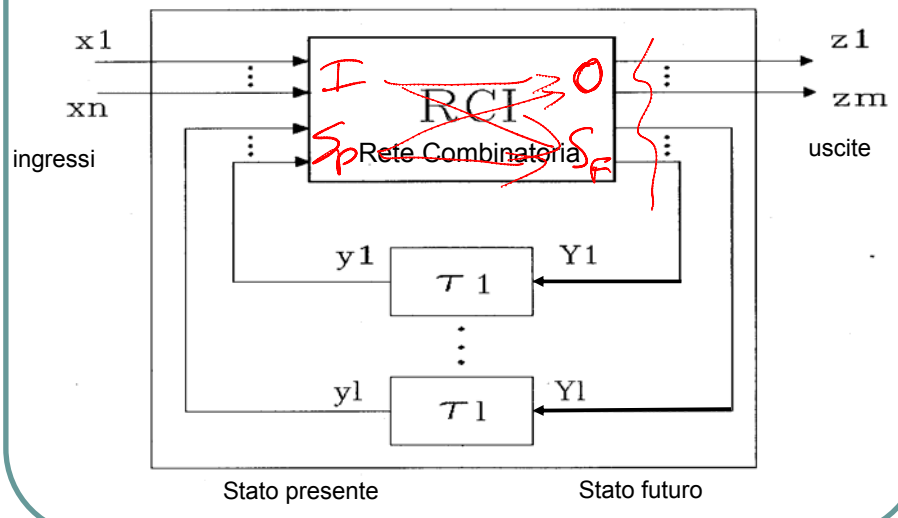
Logica Sequenziale

- La logica sequenziale si distingue da quella combinatoria poiche' in quella sequenziale il valore delle uscite dipende dall'evoluzione temporale degli ingressi che si concretizza nello stato del circuito, come se questo avesse una memoria.
- Circuiti sequenziali si possono descrivere con macchine a stati

Reti Sequenziali

- Nelle **reti combinatorie** l'uscita e' funzione solo degli ingressi. Per una certa configurazione degli ingressi
  I={x1,x2…xn},
  risultano definiti in modo univoco dalla funzione di trasferimento i segnali di uscita
  O={z1,z2,…zm} = f(I)
- Nelle **reti sequenziali** l'uscita viene calcolata anche in base allo stato *S* della rete
  S= {y1, y2, ….yl}.
- Quindi si ha che l'uscita dipende da ingressi e stato
  O=f(I,S),
- mentre lo stato stesso *S=g(I,S passato)*

Modello di rete sequenziale



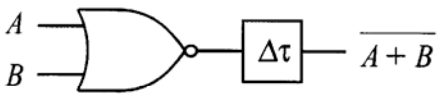
Modello sequenziale

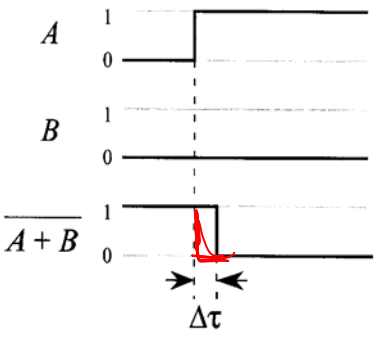
- alfabeto degli ingressi composto da 2^n Combinazioni
- alfabeto delle uscite composto da 2^m Combinazioni
- alfabeto dello stato composto da 2^l Combinazioni

- $O = F(I,Sp)$ $\quad Sp \times I$
- $Sf = G(I,Sp)$ $\quad Sp \times I$

- Sp = Stato passato
- Sf = Stato futuro

Si sfrutta il ritardo delle porte

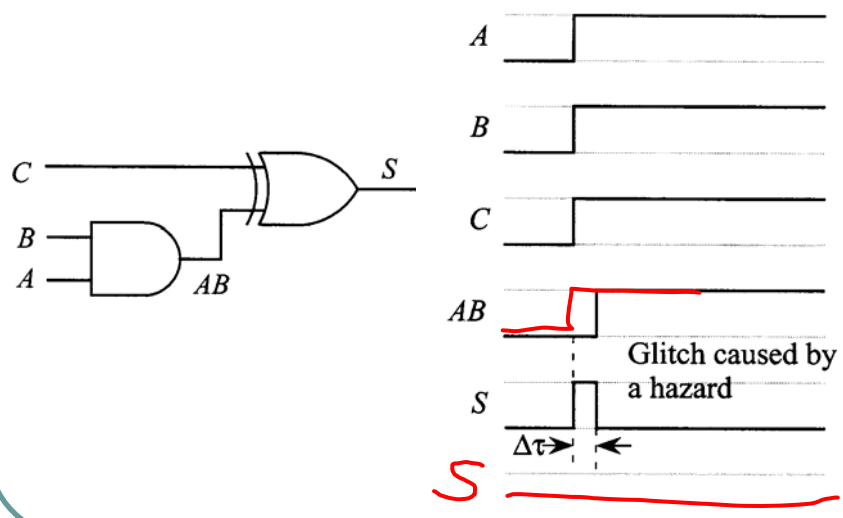




Timing Behavior

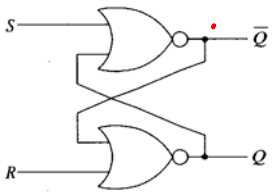
Porta non piu' completamente ideale ma che presenta un ritardo di trasmissione

Effetti indesiderati dei ritardi



Latch di NOR

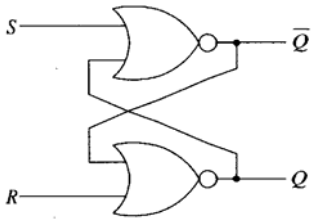
- S : Set (chiudo)
- R : Reset (apro)
- *Qn+1=S+not R Qn*
- Se S,R hanno entrambi valore 1 si ha un uscita non determinabile pertanto viene detto non consentito
- Latch: chiavistello

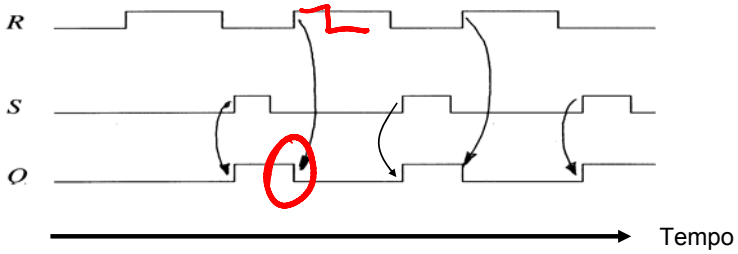


Q_t	S_t	R_t	Q_{t+1}	
0	0	0	0	← non cambia
0	0	1	0	← pone a 0
0	1	0	1	← pone a 1
0	1	1	non consentito	
1	0	0	1	← non cambia
1	0	1	0	← pone a 0
1	1	0	1	← pone a 1
1	1	1	non consentito	

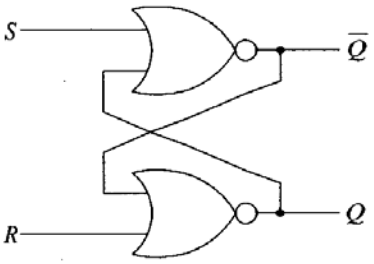
Latch di NOR, rete asincrona

S	R	Q
0	0	non cambia
0	1	0
1	0	1
1	1	non ammesso

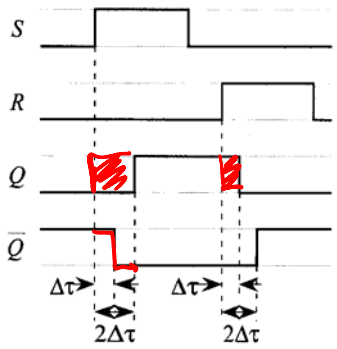




La versione reale, Flip Flop S-R

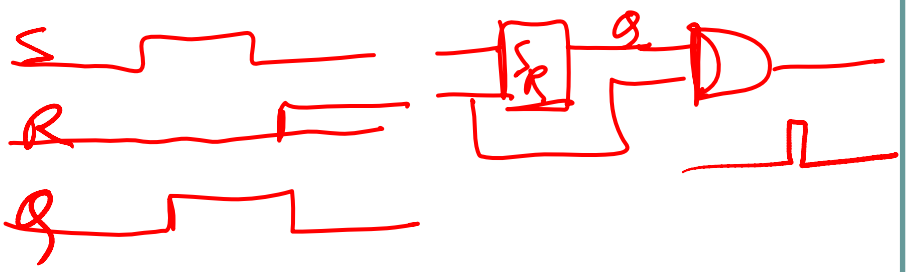


S	R	Q
0	0	non cambia
0	1	0
1	0	1
1	1	non ammesso

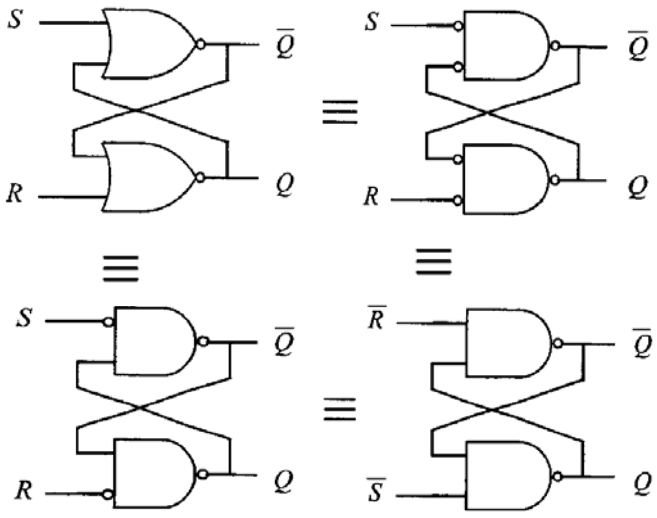


Effetti indesiderati

- Nelle reti sequenziali la presenza di segnali spuri ad o a zero puo- provocare l-arrivo di combinazioni indesiderate ai Flip Flop



IL Latch con altre Porte, tutti equivalenti

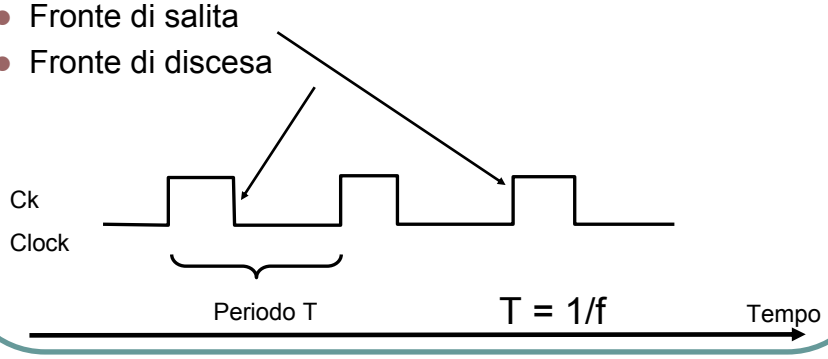


Reti sincrone e asincrone

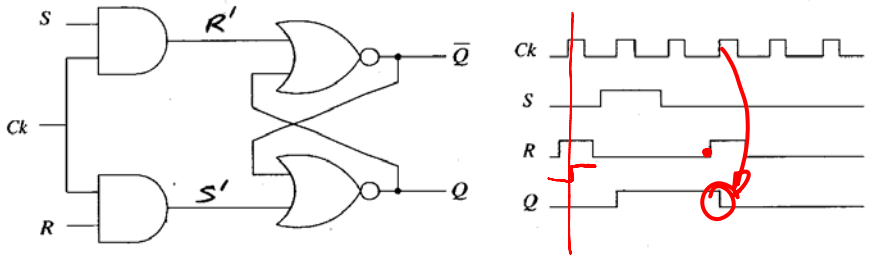
- ***Sequenziali sincrone:***
  - un segnale di clock indica l'istante in cui viene stimato lo stato futuro tramite la funzione *F(I,S)*
- ***Sequenziali asincrone:***
  - la funzione *F(I,S)* viene stimata ad ogni variazione di *S* ed *I*
- Il segnale di **Clock** può essere interpretato in vari modi come vedremo piu' avanti.

IL Segnale di Clock

- T = Periodo di Clock = 1/F
- F = Frequenza di Clock
- 1Mhz di freq. ….. 1 microsecondo di T
- Duty Cycle, rapporto fra il tempo ad 1 e il tempo di T
- Fronte di salita
- Fronte di discesa

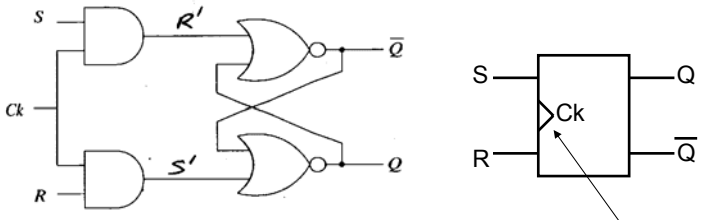


Flip Flop SR (sincrono)



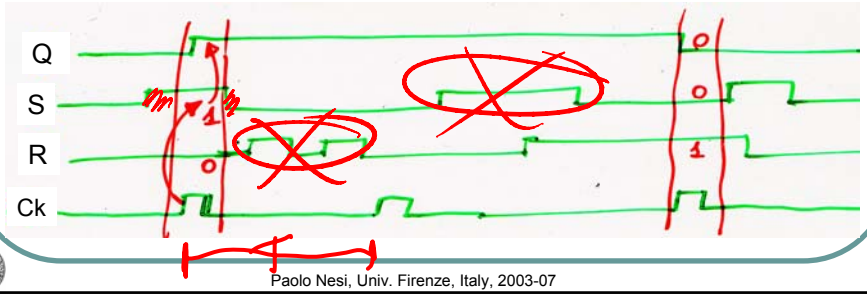
- Nei Flip Flop sincroni il Latch acquisisce gli ingressi e pertanto valuta l'uscita **solo quando lo abilita il segnale di clock**
- Vi possono essere sempre problemi dovuti alla propagazione del segnale e alla realizzazione non ideale delle porte

Flip Flop SR (sincrono)



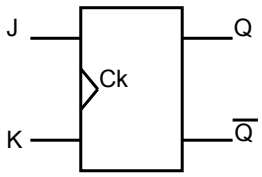
Si attiva sul fronte di salita come specificato dal triangolo

Q (n+1) = S + not R Q(n)



Flip Flop di tipo JK

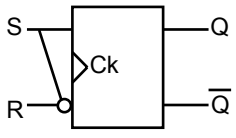
- $Q_{(t+1)} = not\ Q_t\ J + Q_t\ not\ K$

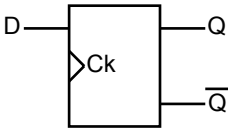


J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	not Q(t)

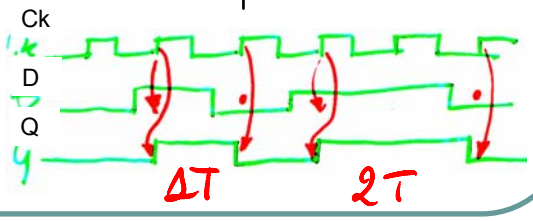
Flip Flop di tipo D, Delay

- *Si ottiene collegando S con un not ad R*
- *Sincronizza/allinea il segnale in ingresso a quello del clock*
- *Realizza un ritardo massimo pari al periodo di clock, Q = D*



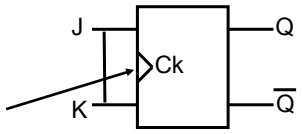


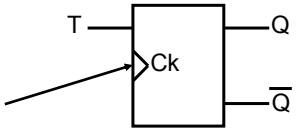
S	R	Q(t+1)
0	0	---
0	1	0
1	0	1
1	1	---



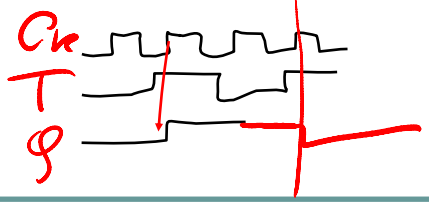
Flip Flop di tipo T, Toggle

- *Commutazione, Q(t+1) = T Q(t)*
- *Si ottiene collegando J con K in un JK*





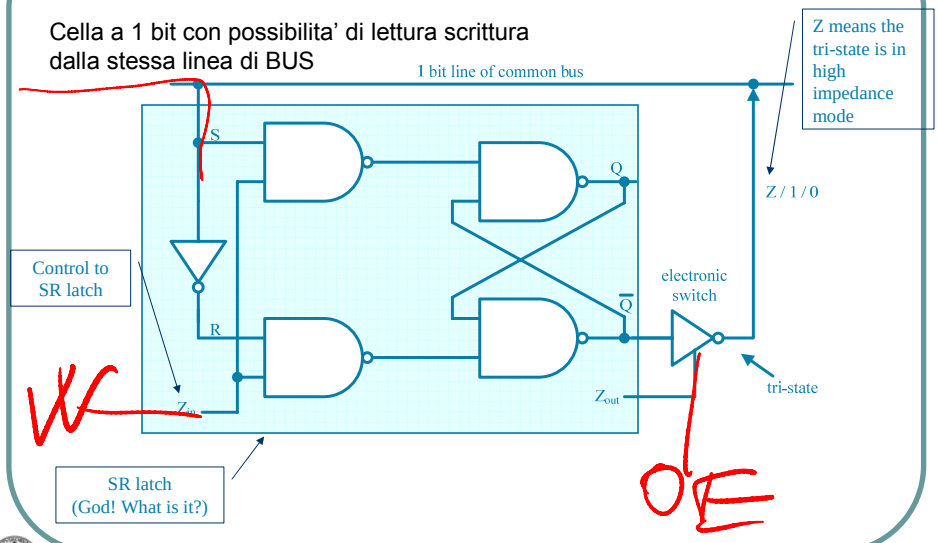
J	K	Q(t+1)
0	0	Q(t)
0	1	----
1	0	----
1	1	not Q(t)



Paolo Nesi, Univ. Firenze, Italy, 2003-07 207

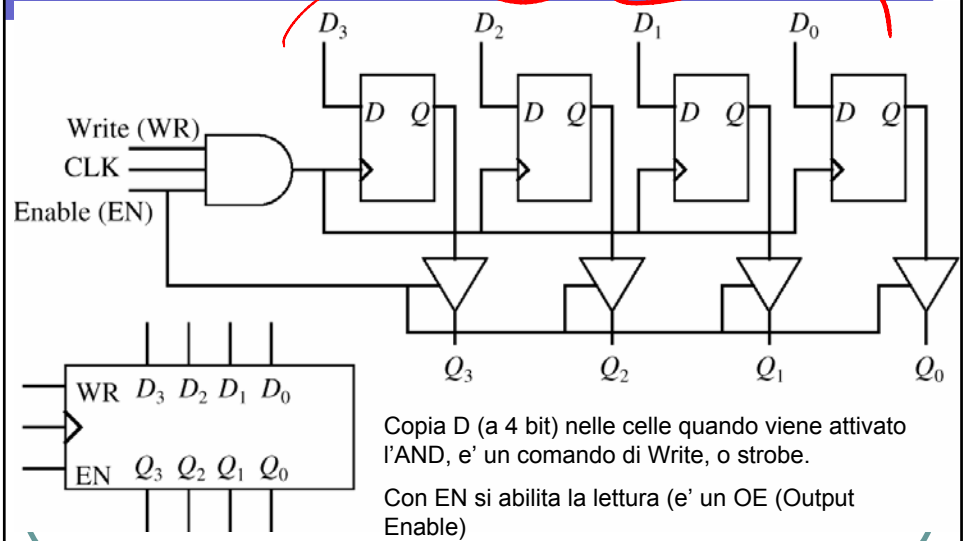
1-bit register with Switch

Cella a 1 bit con possibilita' di lettura scrittura dalla stessa linea di BUS



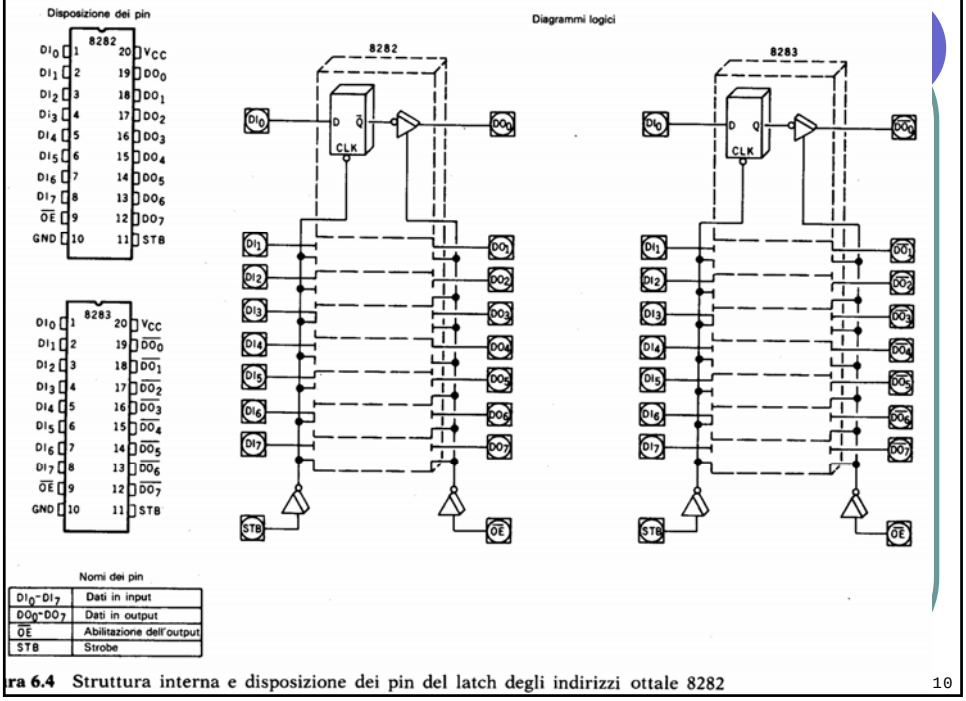
Paolo Nesi, Univ. Firenze, Italy, 2003-07 208

Registro a 4 bit



Copia D (a 4 bit) nelle celle quando viene attivato l'AND, e' un comando di Write, o strobe.

Con EN si abilita la lettura (e' un OE (Output Enable)

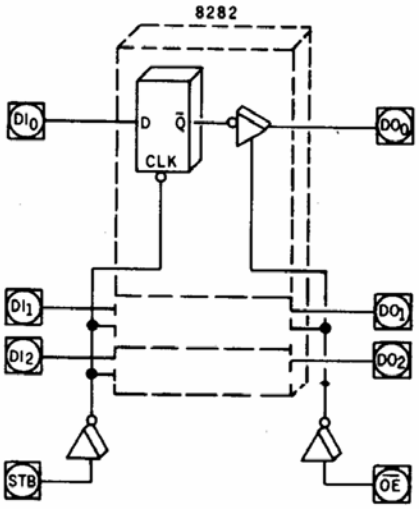


ra 6.4 Struttura interna e disposizione dei pin del latch degli indirizzi ottale 8282

8282

Comando di STB, Strobe per copiare l'informazione sull'ingresso di D in D stesso.

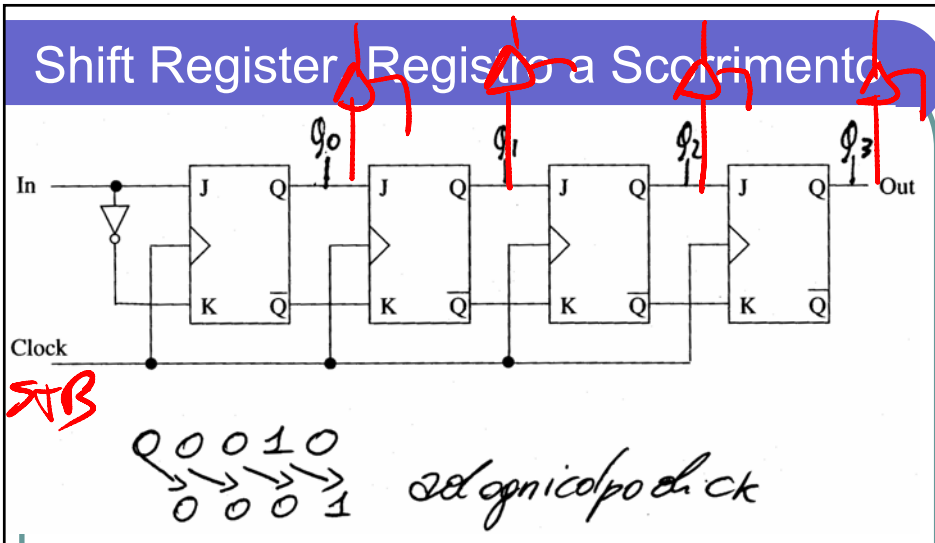
Comando notOE per abilitare l'uscita, Output Enable.



Nomi dei pin

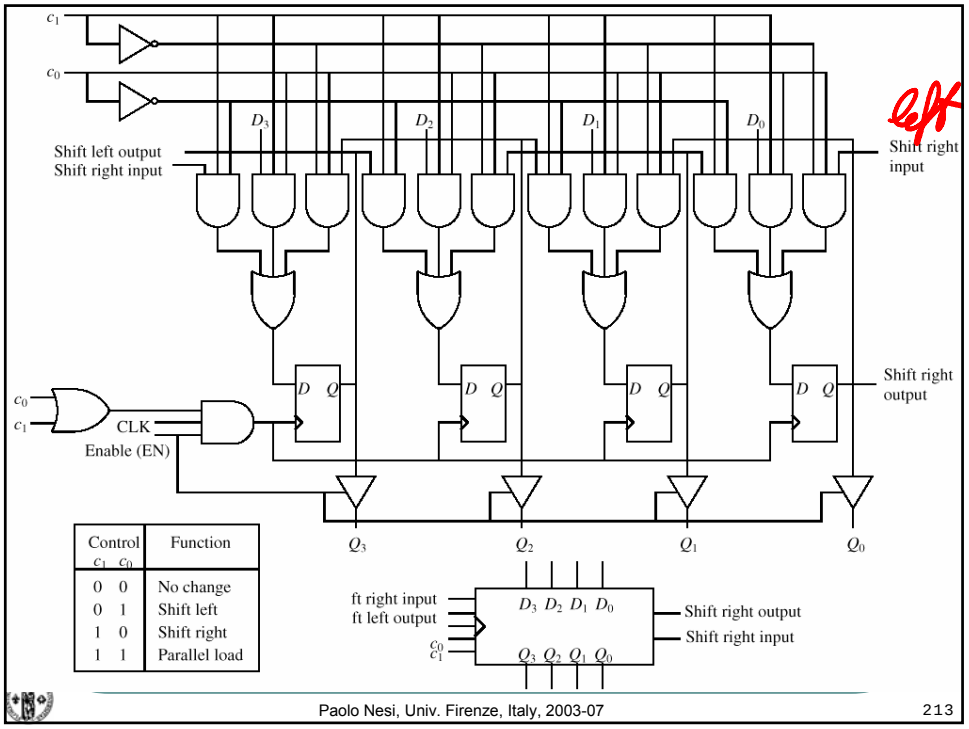
DI_0-DI_7	Dati in input
DO_0-DO_7	Dati in output
$\overline{OE}$	Abilitazione dell'output
STB	Strobe

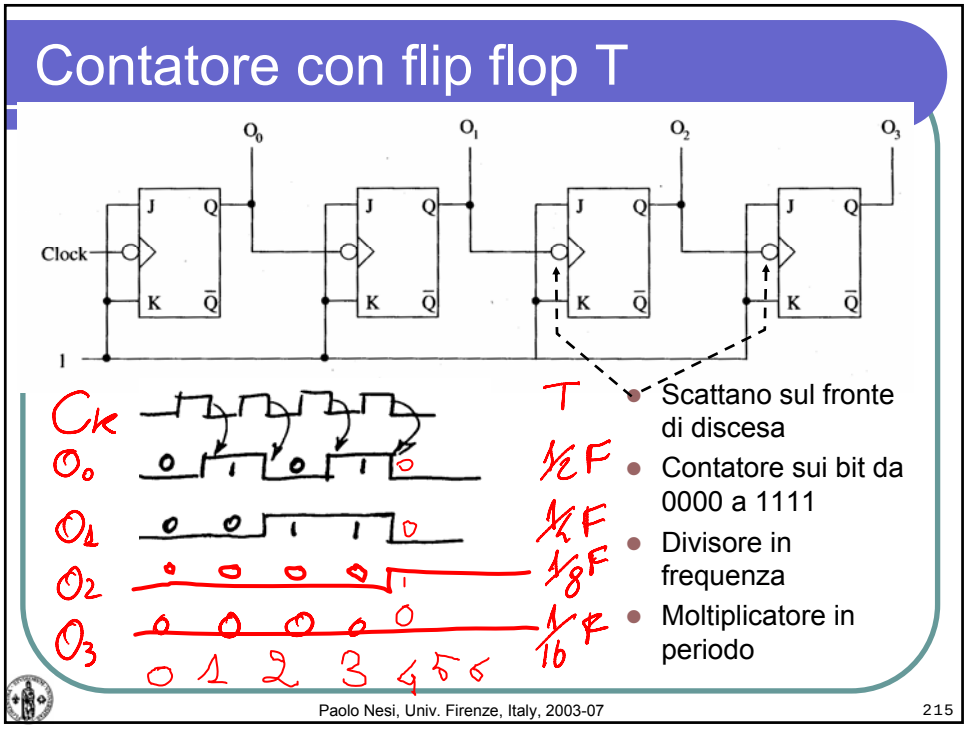
Shift Register, Registro a Scorrimento



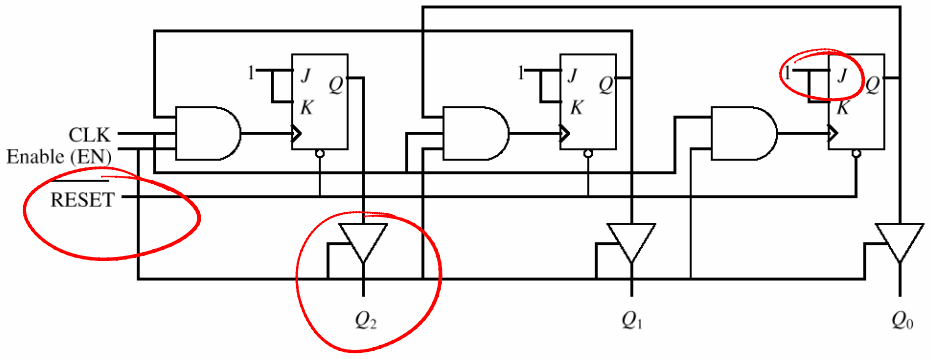
Il registro scorre la sequenza che arriva in IN, ad ogni colpo si scrorre di un bit.

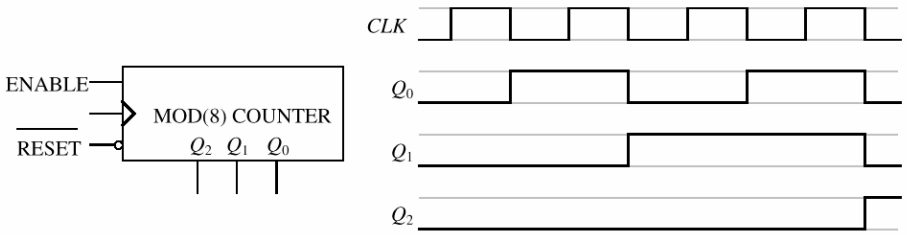
Clock e' in effetti uno Strobe (STR, un'attivazione) che permette di Scorrere di uno


c_1
c_0
D_3
D_2
D_1
D_0
Shift left output
Shift right input
Shift right input
c_0
c_1
CLK
Enable (EN)
D Q
Shift right output
Q_3
Q_2
Q_1
Q_0
Control c_1 c_0
Function
0 0 No change
0 1 Shift left
1 0 Shift right
1 1 Parallel load
ft right input
ft left output
D_3 D_2 D_1 D_0
Q_3 Q_2 Q_1 Q_0
Shift right output
Shift right input
Paolo Nesi, Univ. Firenze, Italy, 2003-07
213


Contatore con flip flop T
Q_0
Q_1
Q_2
Q_3
Clock
J Q
K $\bar{Q}$
1
Ck
Q_0
Q_1
Q_2
Q_3
0 1 2 3 4 5 6
T
• Scattano sul fronte di discesa
• Contatore sui bit da 0000 a 1111
• Divisore in frequenza
• Moltiplicatore in periodo
Paolo Nesi, Univ. Firenze, Italy, 2003-07
215

Contatore con JK(T) modulo 8

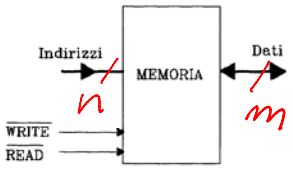


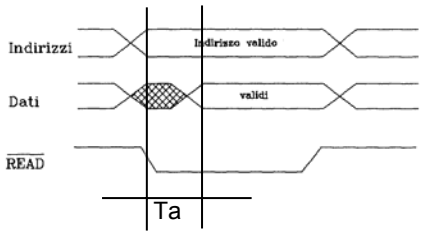


6

La Memoria, modello generale

- **Bit di Indirizzamento, Bus Indirizzi**
- **Bit dati, Bus dati**
- **Primo concetto di BUS**
- **Segnali di Controllo, Read, Write, Select…., Bus Controlli**
- **Tempo di lettura**

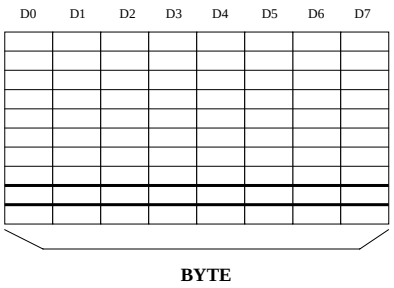




Paolo Nesi, Univ. Firenze, Italy, 2003-07

218

Rappresentazione logica della Memoria



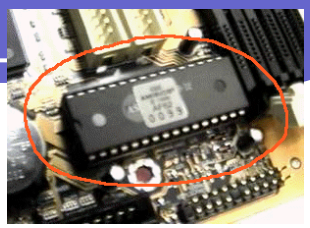
Vettore composto da celle in cui vengono immagazzinate informazioni.
Ogni cella ha un indice che costituisce il suo "indirizzo" all'interno della memoria.
Dati e informazioni contenuti sono in forma binaria.

La dimensione di una cella è data dal numero di bit che essa contiene: può essere di un bit, ma anche di 8, 16, 32, 64.

Tipi di Memoria

- Memorie a stato solido
  - RAM, Random Access Memory
    - SRAM, DRAM, etc.
  - ROM, Read Only Memory
  - PROM, Programmable ROM
  - EPROM, Erasable PROM
  - EEPROM, Electrically Erasable PROM
  - Flash…….
- Esistono anche Memorie su supporto magnetico
  - Dischi, nastri, HD, FD, etc.

La Memoria ROM



ROM, Read Only Memory
(memoria di sola lettura)

I dati immagazzinati nella memoria ROM permangono sempre anche se viene a mancare l'alimentazione.

Contengono le istruzioni di base (firmware, BIOS) per consentire l'avvio: l'inizializzazione delle periferiche del calcolatore, il collegamento della stessa con i terminali di ingresso /uscita.

RAM, Random Access Memory

memoria di lettura scrittura.

conserva i dati intermedi delle elaborazioni, i risultati dei calcoli, ecc.

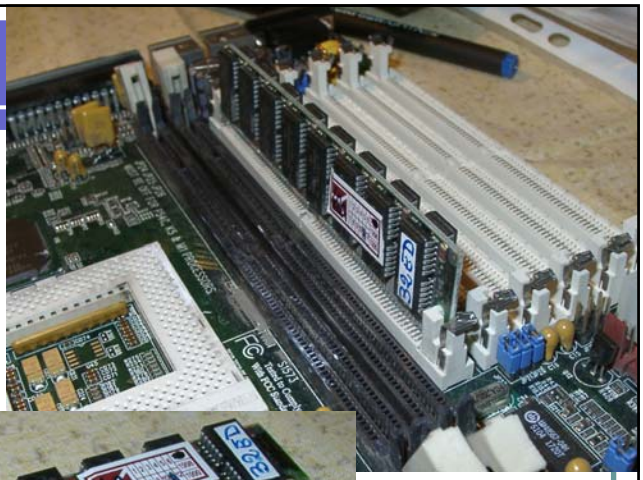
volatile, si perde l'informazione se viene a mancare l'alimentazione.

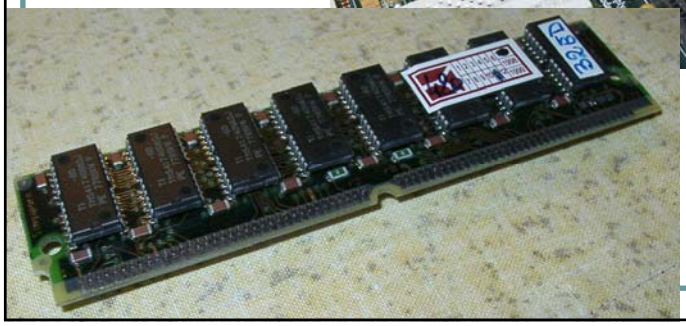
Si chiama ad accesso casuale poiché è possibile accedere ad ogni singola cella in modo diretto e pertanto la modalità casuale e possibile.

Questa modalità è da vedere contrapposta alla modalità sequenziale per la quale per accedere ad un cella si deve prima avere fatto accesso alla precedente, etc. Partendo dalla prima cella della memoria.

Memorie SIM

- 256 Mw a16
- 8 Chip da un Bit
- BA: ?? bit
- BD: ?? bit

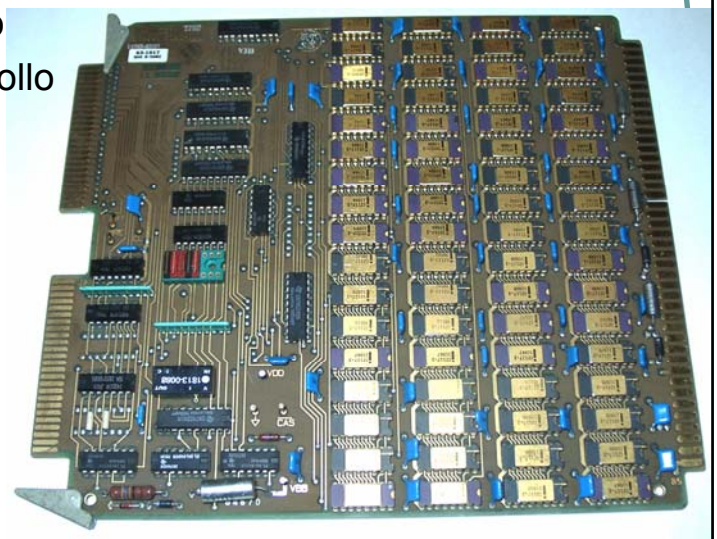




223

Una scheda di Memoria piu' datata

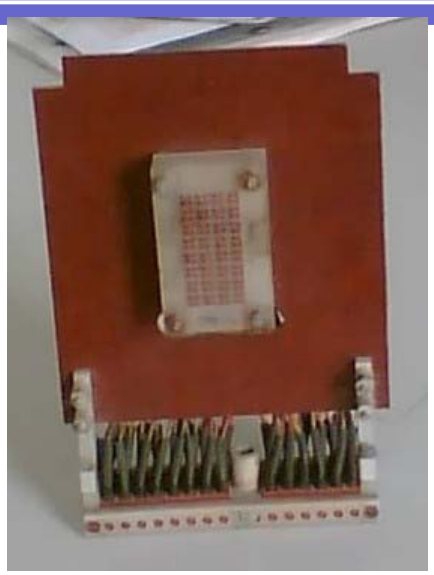
- 16 bit/chip
- 1 di Controllo



Paolo Nesi, Univ. Firenze, Italy, 2003-07

224

Una scheda di Memoria piu' datata



Altri tipi di Memoria

PROM (Programmable ROM)
Può essere programmata dal produttore una sola volta.
La programmazione di queste memorie avviene tramite un dispositivo elettronico detto programmatore che produce particolari tensioni di alimentazione e sequenze di operazioni al fine di provocare delle "lesioni" permanenti nella matrice che rappresenta la memoria.

EPROM (Erasable PROM)
Questa memoria può essere scritta e cancellata, ma solo globalmente sottoponendola a raggi UV per un certo numero di minuti. Dopo la cancellazione può essere riscritta.
Questa operazione può essere ripetuta un numero limitato di volte. Si noti che per la cancellazione è necessario smontare la EPROM dal circuito sul quale è installata.

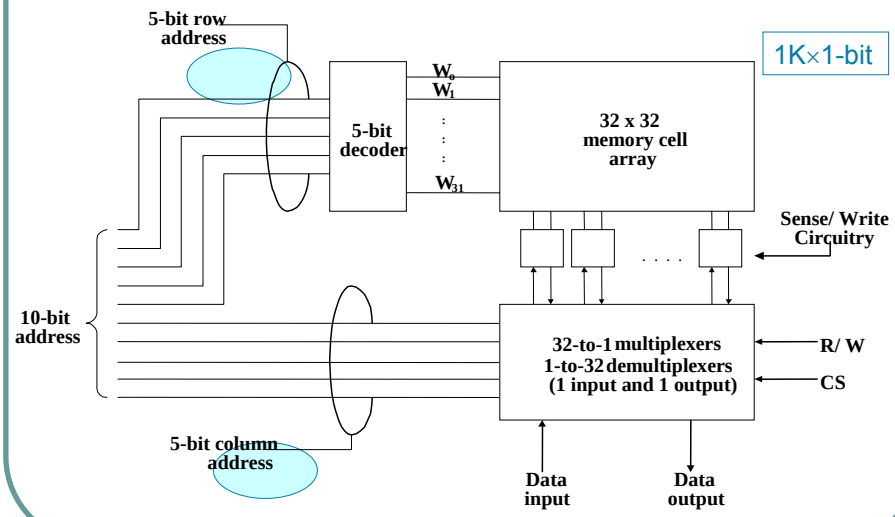
Altri tipi di Memoria

EEPROM (Elettrically Erasable PROM)
Come la EPROM ma le operazioni di cancellazione possono essere più selettive e vengono effettuate tramite segnali elettrici. Non è necessario rimuove la EEPROM dal circuito per effettuare le cancellazioni. Dopo la cancellazione può essere riscritta. Questa operazione può essere effettuata un numero limitato di volte.

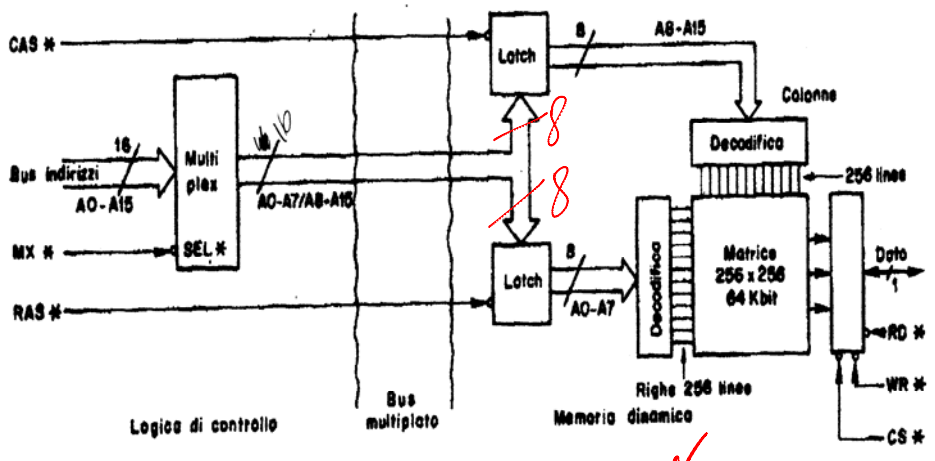
RAM, ROM, EPROM, PROM, EEPROM sono memorie cui si accede direttamente in base all'indirizzo della cella

Nei calcolatori viene utilizzata solitamente memoria RAM per l'esecuzione di programmi e contenere dati, mentre EPROM viene utilizzata come memoria ROM (per es. per il BIOS).

Un chip di memoria, 1Kbit

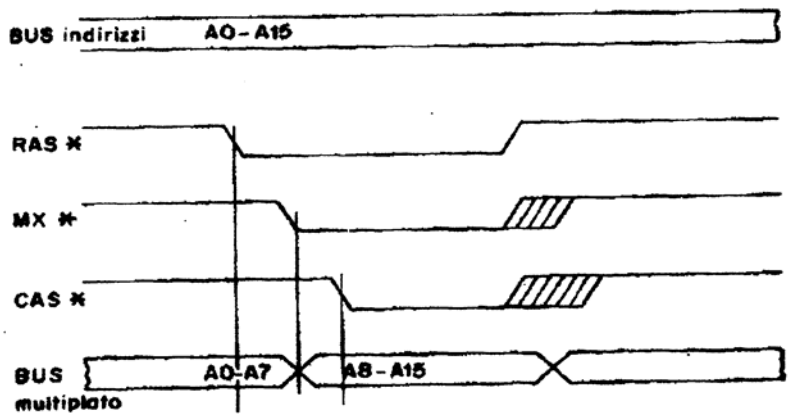


anatomia di una Memoria DRAM



- Intacciamento con memorie DRAM.

Diagramma di accesso alla memoria



- Diagrammi temporali del colloquio con memorie DRAM.

Memorie a stato solido

- Con N segnali di indirizzamento si ha la possibilità di indirizzare 2^N diverse gruppi di celle
- Ogni gruppo di celle puo' essere di *M* bit, Per esempio 8, 16, o 32 (le taglie più probabili)
- Le memorie si possono comporre utilizzando il CS (chip select) per raggiungere dimensioni maggiori
- Sui tempi di accesso e la composizione ritorneremo quando le vedremo dentro l'elaboratore.

Calcolatori Elettronici

CDL in Ingegneria Elettronica

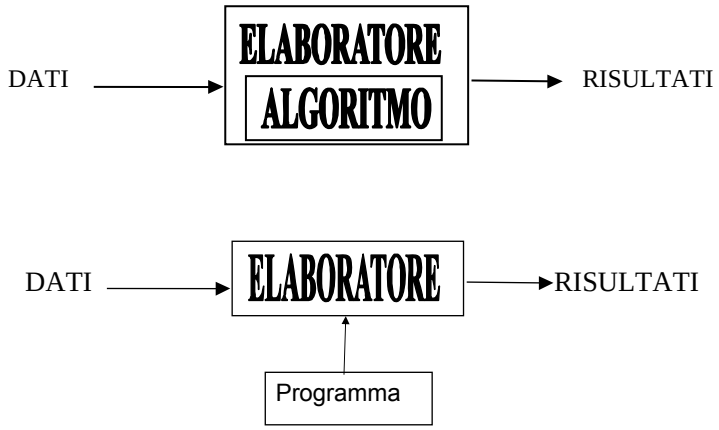
Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 4, Le architetture degli elaboratori

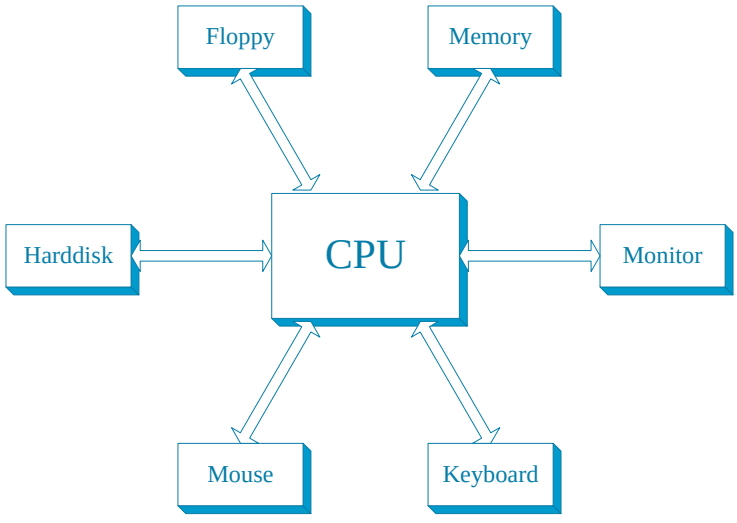
Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2007

L'elaboratore



Paolo Nesi, Univ. Firenze, Italy, 2003-07 233

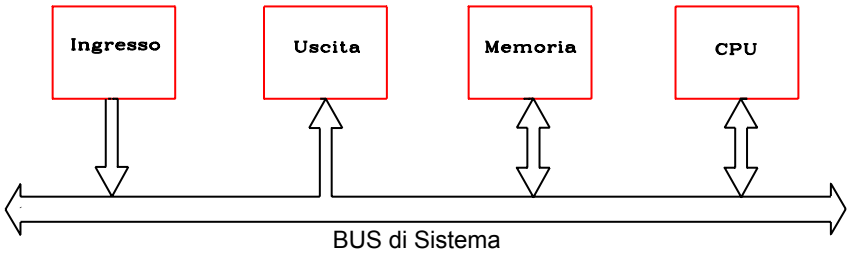
CPU e relazioni funzionali !



Paolo Nesi, Univ. Firenze, Italy, 2003-07 234

Schema di riferimento

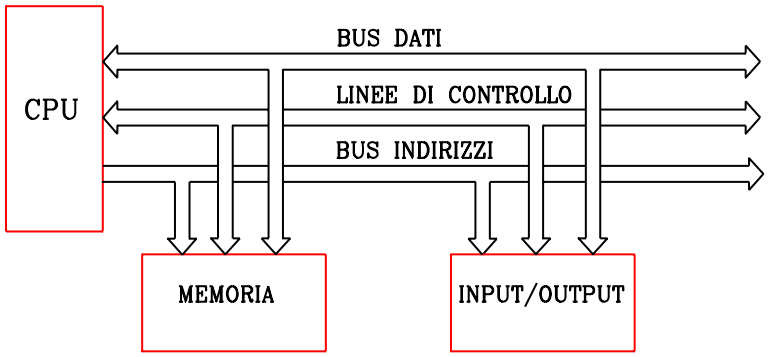
- Per Il momento lo schema di riferimento sarà quello di figura
- Corrisponde allo schema dei PC anni 80
- Tuttora in largo uso nei sistemi di controllo
- Le unità sono collegate da un bus di sistema



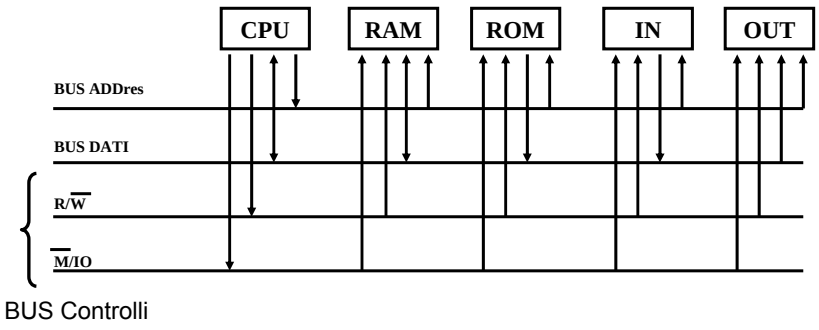
Alcuni Cenni alle periferiche

- **I/O, Input/Output, Ingressi/Uscita**
  - Input: keyboard (tastiera), mouse, scanner
  - Output: monitor, printer
- **Memory, Memoria**
  - Di base: RAM, ROM, stato solido….
  - Di Massa: dischi (HD, FD), nastri,
- **CPU (Central Processing Unit), microprocessore**
  - ALU,
  - unità di controllo,
  - registri,
  - etc.

La Struttura del BUS di Sistema

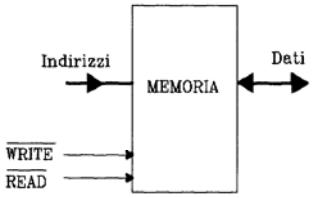


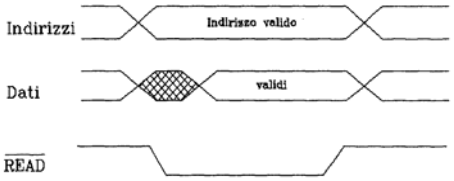
Direzione dei Segnali sul BUS



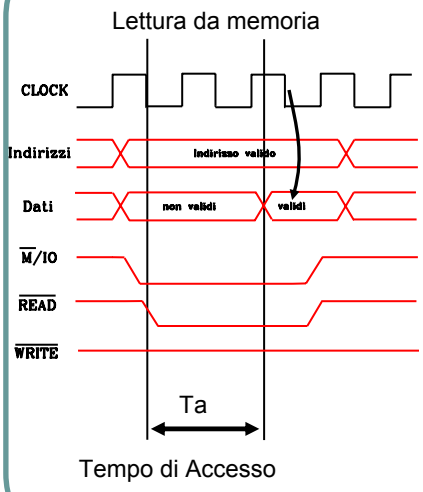
La Memoria e l'elaboratore

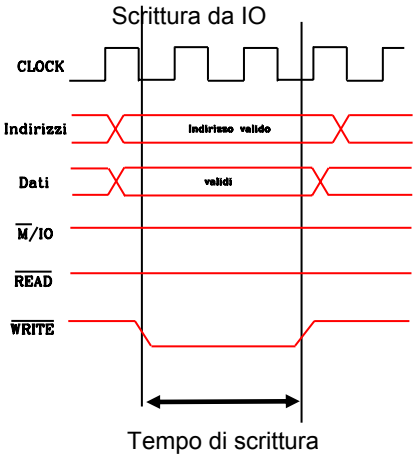
- Bus Indirizzi: Bit di Indirizzamento, N bit
- Bus Dati, M bit
- Bus Controlli Segnali di Controllo, Read, Write, Select….
- Tempo di lettura, tempo di accesso
- Si ha una capacita di $M*2^N$ bit





Transazioni sul bus, Ciclo macchina



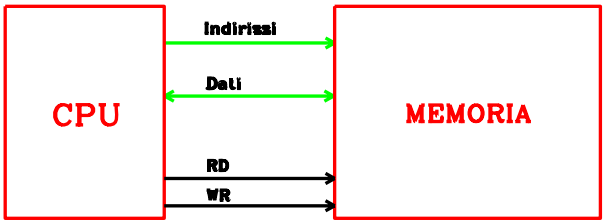


Selezione, Memoria/IO R/W

Not M/IO	not R	Not W	Azione
0	0	0	----none
0	0	1	Read Memory
0	1	0	Write Memory
0	1	1	----impossibile
1	0	0	----none
1	0	1	Read IO
1	1	0	Write IO
1	1	1	----impossibile

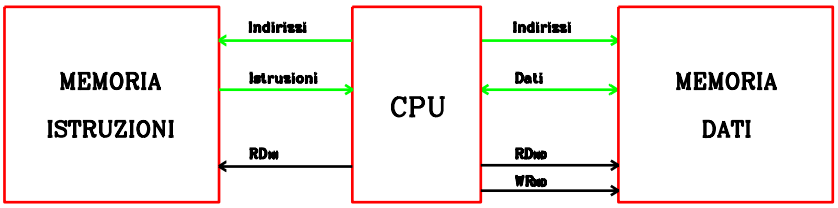
Architettura di Von Neuman

- memoria indifferenziata per dati o istruzioni, solo l'interpretazione da parte di CPU stabilisce se una data configurazione di bit è da riguardarsi come un dato o come un'istruzione



Architettura Harward

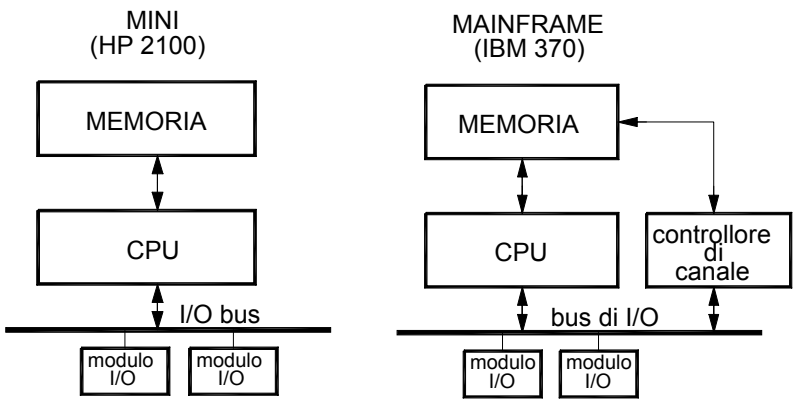
- Due memorie distinte: la memoria istruzioni e la memoria dati.
- Il comando di lettura della memoria istruzioni è superfluo, in quanto si può immaginare che questa memoria sia sempre e soltanto letta



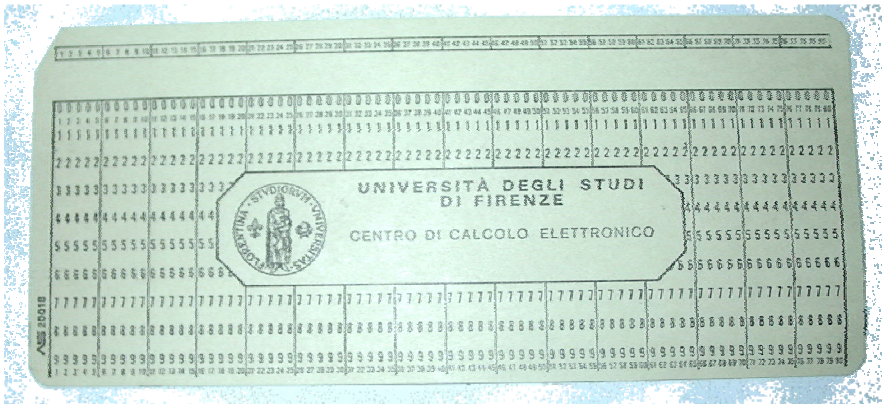
Confronto

- **Von Neuman**
  - Accesso a istruzioni e dati nella stessa memoria
  - Flessibilità nello sfruttamento della memoria
  - Rischio di manipolazione del codice
  - Minore costi di realizzazione
- **Harward**
  - Robustezza alla manipolazione del codice
  - Accesso contemporaneo a codice e dati
  - Costi maggiori di realizzazione
  - Minore flessibilità

Struttura (anni 70)



Schede per Lettori di Schede



Un Micro degli anni 70, 4004, 4 bit

intel MCS-4 MICRO COMPUTER SET

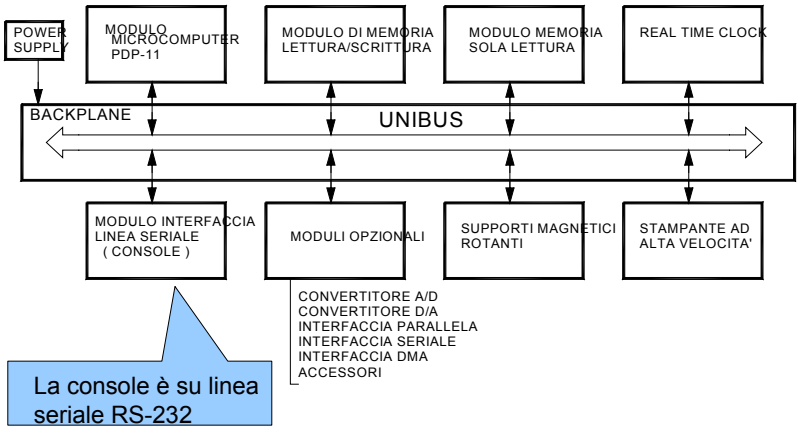
NOVEMBER 1971

- Microprogrammable General Purpose Computer Set
- 4-Bit Parallel CPU With 45 Instructions
- Instruction Set Includes Conditional Branching, Jump to Subroutine and Indirect Fetching
- Binary and Decimal Arithmetic Modes
- Addition of Two 8-Digit Numbers in 850 Microseconds
- 2-Phase Dynamic Operation
- 10.8 Microsecond Instruction Cycle
- Easy Expansion—One CPU can Directly Drive up to 32,768 Bits of ROM and up to 5120 Bits of RAM
- Unlimited Number of Output Lines
- Single Power Supply Operation (V_{DD} = —15 Volts)
- Packaged in 16-Pin Dual In-Line Configuration

http://smithsonianchips.si.edu/ice/4004.htm

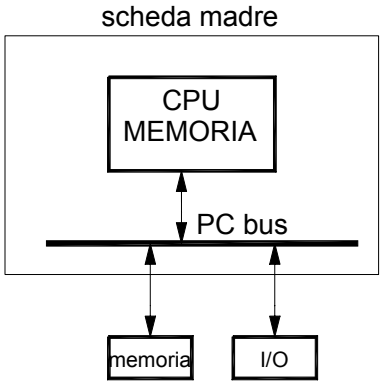
Paolo Nesi, Univ. Firenze, Italy, 2003-07 248

Struttura PDP-11 (anni 70-80)



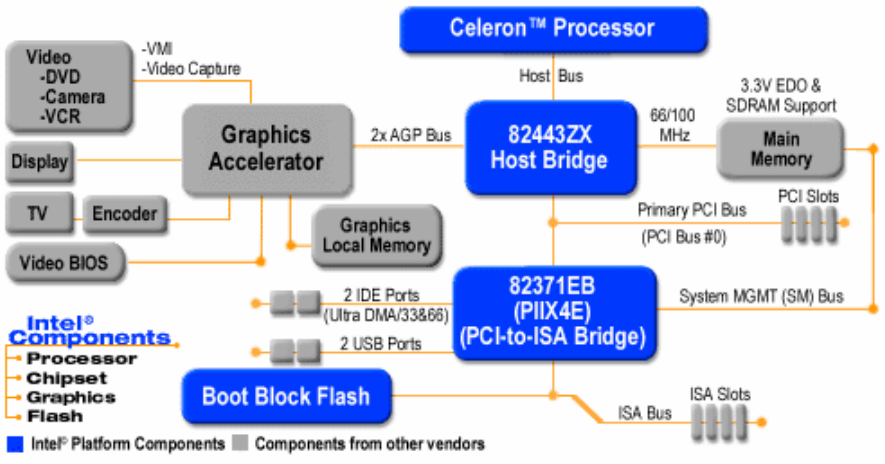
Paolo Nesi, Univ. Firenze, Italy, 2003-07 249

Struttura PC (anni 80)



- E' sparita la console RS232
- Il Video e la tastiera sono collegati direttamente alla scheda madre.
- Per il video c'è una RAM apposita
- Memoria di espansione e periferici sono su schede collegate sul bus

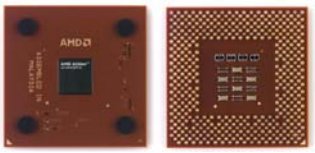
Struttura PC corrente



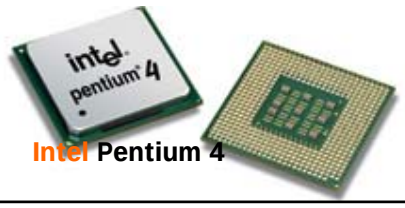
Microprocessore

Un microprocessore o CPU è un dispositivo elettronico ad elevata integrazione, che può svolgere operazioni aritmetiche, logiche e di controllo, su dati generati dal microprocessore stesso o forniti dall'esterno.

Il microprocessore per funzionare, deve essere inserito in una struttura in grado di utilizzarne le potenzialità, fornendo al dispositivo, attraverso programmi definiti dall'utente, le informazioni necessarie, relative al tipo di operazione da eseguire ed i dati su cui operare.

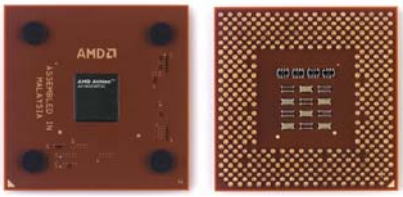


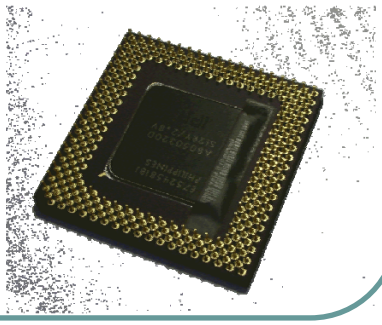
AMD Athlon XP



Intel Pentium 4

CPU



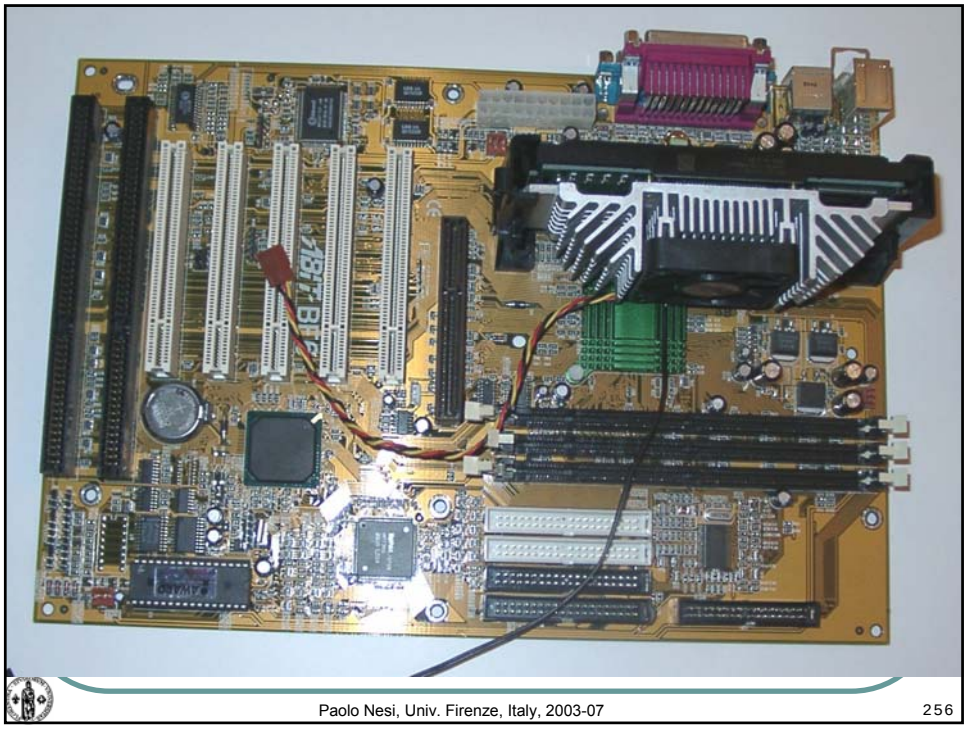


CPU

Paolo Nesi, Univ. Firenze, Italy, 2003-07 254

CPU

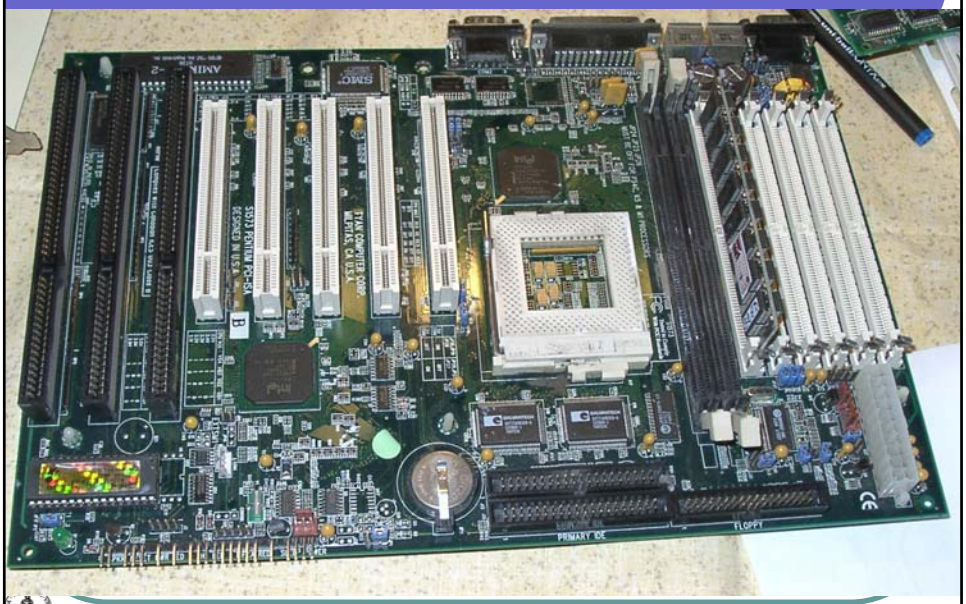
Paolo Nesi, Univ. Firenze, Italy, 2003-07 255



Scheda Madre precedente

- Si noti:
  - La CPU: pentium a tecnologia ibrida
  - I BUS: EISA, PCI, Local Bus pe rscheda video,
  - 3 Slot per la memoria
  - Quarzo per il Clock
  - Connettori per IO controllori di disco
  - Connettori/zoccoli per IO porta parallela, seriale, tastiera
  - etc.

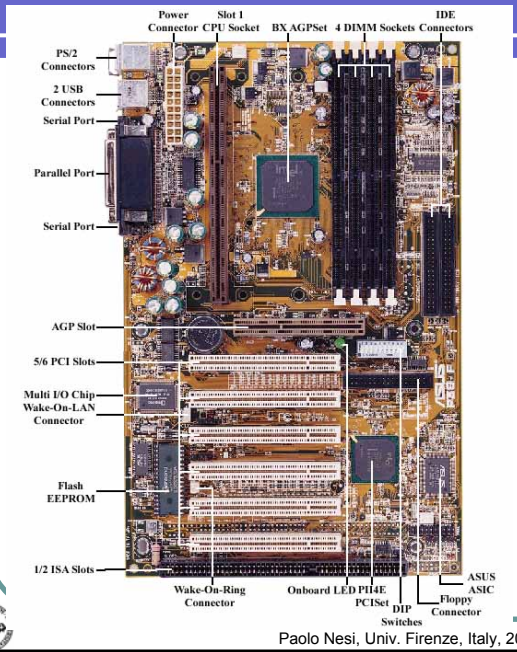
Scheda Madre

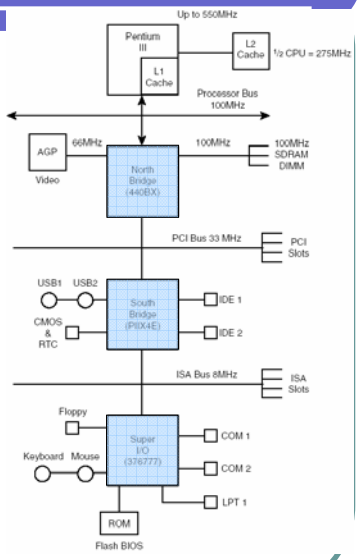


La scheda madre precedente

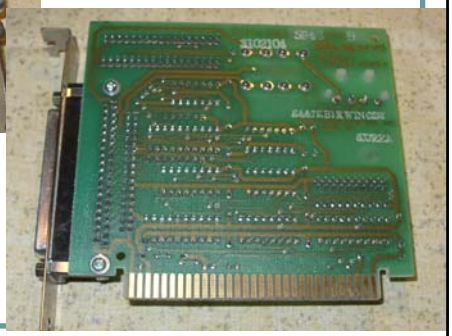
- Si noti:
  - Zoccolo per la CPU
  - I BUS: EISA, PCI,
  - Slot per la memoria: due tipi diversi
  - Quarzo per il Clock
  - Connettori per IO controllori di disco
  - Connettori/zoccoli per IO porta parallela, seriale, tastiera etc.

La scheda madre



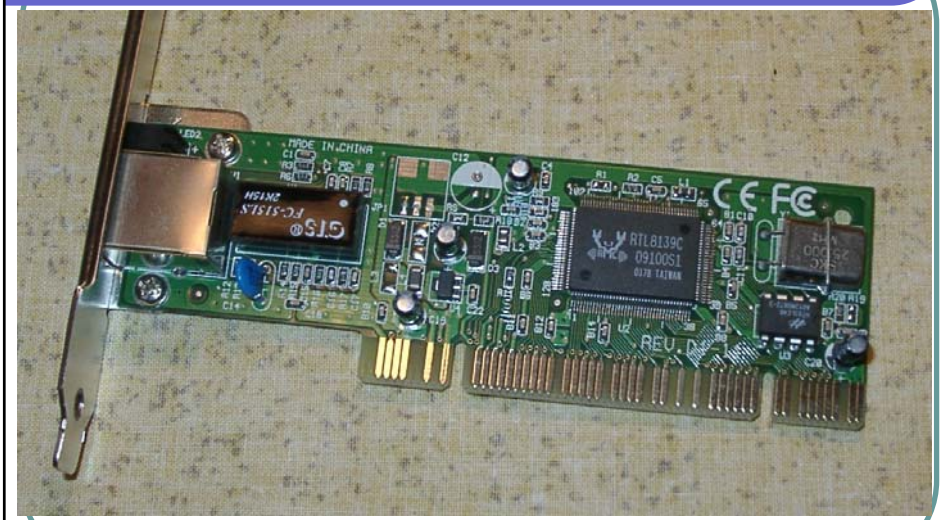


Standard ISA Board (controllore)



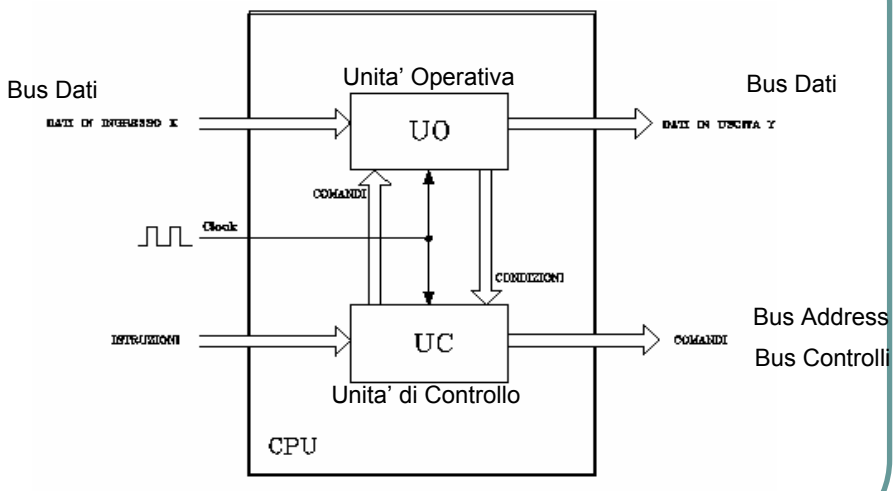
EISA Board (controllore SCSI)

PCI Board (Scheda di Rete)



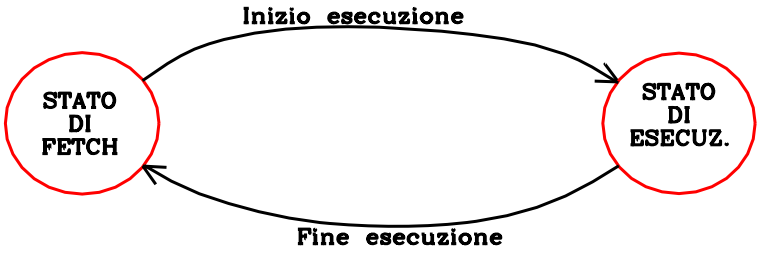
Mother board with boards

Schema aggregato della CPU

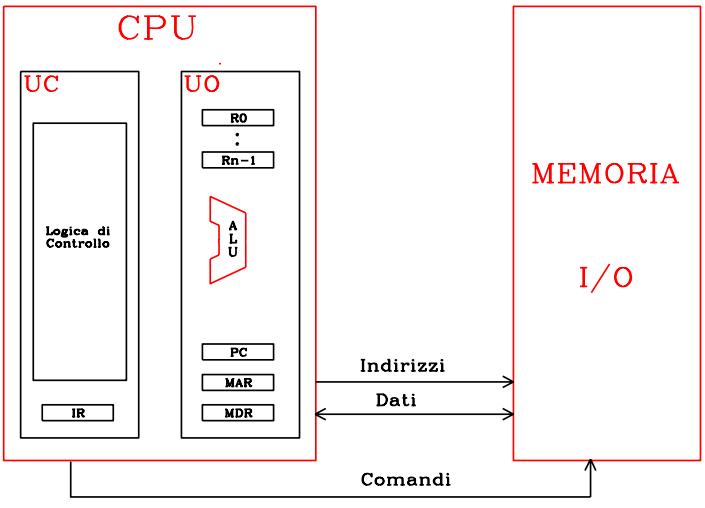


Fetch-Esecuzione

- **Fetch:** Prelievo e decodifica dell'istruzione
  - Fase comune a tutte le istruzioni
- **Esecuzione:** Fase in cui vengono eseguite le azioni previste dal codice di operazione
  - Fase diversa da istruzione a istruzione



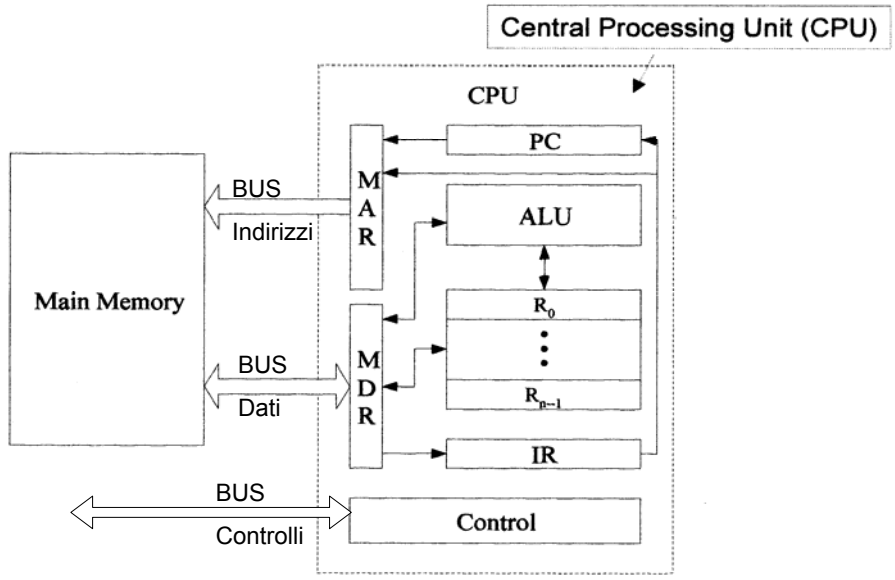
Componenti essenziali



La Gerarchia delle Memorie

- I registri interni sono delle memorie molto veloci ai quali si fa riferimento direttamente dalla ALU
  - tempi di accesso dell'ordine di 10 nsec
- La struttura della Memoria a stato solido e' stata già presentata e discussa in precedenza
- RAM
  - più lenta dei registri interni
  - velocità dell'ordine dei 40 – 80 nsec, nanosecondi
  - DRAM, RAM
  - per i programmi applicativi
- ROM
  - per garantire il funzionamento all'accensione,
  - contiene le prime istruzioni che vengono eseguite all'accensione

Maggiore Dettaglio relaz. Mem-CPU



Registri di CPU

- **MAR: Memory Address Register**
  - contiene l'indirizzo della locazione di memoria da leggere o scrivere.
  - La dimensione del MAR determina l'ampiezza dello spazio di memoria fisica; dalla fine degli anni '80 vengono prodotti microprocessori con bus indirizzi a 32 bit e MAR a 32 bit
- **MDR: Memory Data Register**
  - registro attraverso il quale viene scambiata l'informazione tra la memoria e la CPU
  - Tradizionalmente la dimensione dell MDR dà la misura del grado di parallelismo della macchina (8, 16, 32, 64 bit)
- **R0, R1,...Rn: registri di uso generale**
  - Registri di uso generale,
  - elevata velocità,
  - memorizzazione temporanea dei dati,
  - realizzati come serie di FF

Elementi delle CPU

- **UC: Unità di Controllo, Control Unit**
  - Decodifica le istruzioni contenute nell'IR e genera i segnali di controllo
  - Controlla le altre unità al fine di completare l'istruzione presente in IR
  - Produce i segnali che escono dalla CPU, verso il BUS controlli
  - Legge alcuni segnali che entrano nella CPU, per esempio il Clock…..e dal BUS controlli
- **UO: Unità operativa**
  - Contiene i registri
  - Contiene la ALU

Registri di CPU

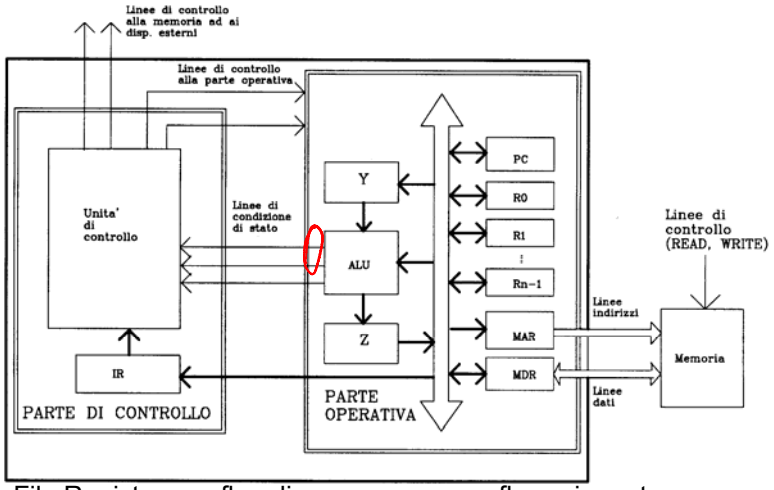
- **IR: Instruction Register**
  - Usato per contenere l'istruzione in corso di esecuzione.
  - Caricato in fase di fetch dalla memoria prendendo il contenuto della cella identificata come indirizzo dal PC
  - Rappresenta l'ingresso che determina le azioni svolte durante la fase di esecuzione.
  - Dall'IR viene decodificata l'istruzione
- **PC: Program Counter**
  - Tiene traccia dell'esecuzione del programma
  - Contiene l'indirizzo di memoria della prossima istruzione
  - Viene aggiornato per indirizzare l'istruzione successiva o parti di questa

Esempio di sequenza di istruzioni

- M[3]=M[1]+M[2]

R6 ← M[1]	6 periodi (colpi) di Clock
R7 ← M[2]	6 Clock
R8 = R6 + R7	3 ck
M[3] ← R8	6 ck
	21ck

Maggiore Dettaglio vicino alla ALU

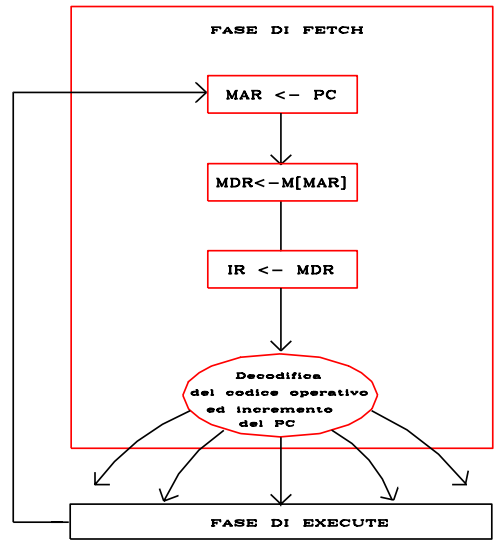


- + File Register con flag di: zero, carry, overflow, sign, etc., per mantenere lo stato della ALU
- BUS Interno

ALU, Arithmetic Logic Unit

- Esegue operazioni aritmetiche e logiche
- I risultati sono scritti nei registri o in memoria
- I dati di partenza sono letti dai registri o dalla memoria
- I due precedenti punti possono avere delle restrizioni dipendentemente dalla architettura per esempio architetture con registro accumulatore, etc.

Fase di Fetch



MAR ← PC

Not READ

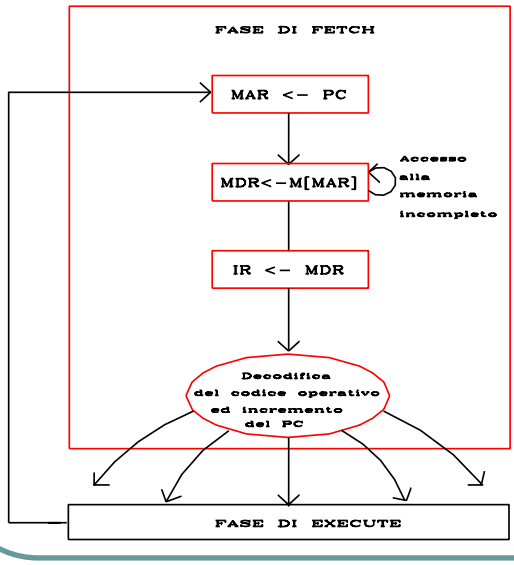
MDR ← M[MAR]

IR ← MDR

Decodifica dell'isttruz.

PC ← PC + 1

Fase di Fetch, IST multibyte



MAR ← PC

Not READ

MDR ← M[MAR]

IR ← MDR

Decodifica dell'istruz.

PC ← PC + 1

……

PC ← PC + 1

Su più letture per completare la lettura dell'istruzione che può essere su piu' byte o word…..

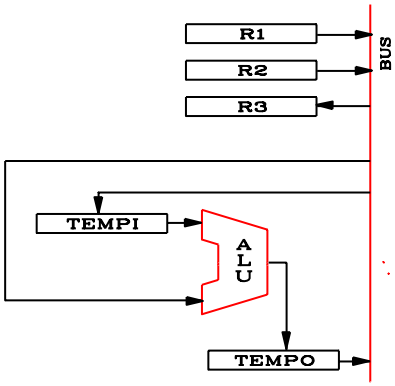
L'Esecuzione dell'istruzione

- **Fetch dell'istruzione,** IF
  - Fetch per come e' stato visto
- **Decodifica dell'istruzione**, decode, ID
  - Decodifica dipendentemente dal tipo di codifica
- **Esecuzione** dell'istruzione, execution, EXE
  - Può richiedere l'uso della ALU o azioni di gestione generale della CPU
  - Può implicare la lettura da memoria
- **Scrittura dei Risultati,** write back, WB
  - Scrittura dei risultati in memoria o su I/O

A che punto siamo, Cosa vediamo ora

- Abbiamo visto
  - Instruction fetch, l'acquisizione delle istruzioni
- Vediamo ora
  - l'esecuzione delle istruzioni
- In seguito vedremo la
  - decodifica delle istruzioni

Data Path - 1 bus



Esecuzione in più passi.
Esempio: **ADD R3,R1,R2**
"Cioe' R3=R1+R2"

Richiede almeno **tre periodi di clock**:

- R1_out; TEMPI_in
- R2_out; TEMPI_out; ADD; TEMPO_in
- TEMPO_out; R3_in

Segnali per buffer Tri-State

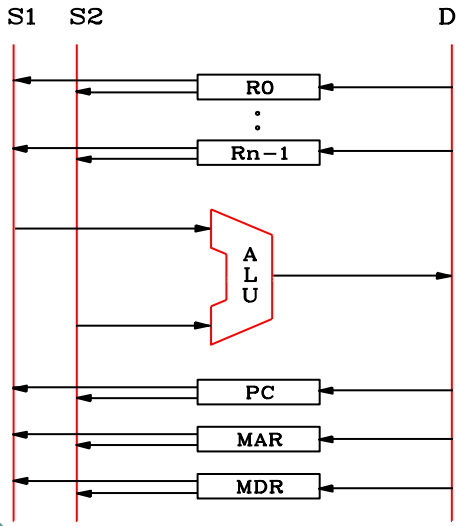
Decodifica dell'istruzione, idea

- Supponiamo che per ogni fase si possano fare 16 cose diverse (per esempio comandare 16 segnali diversi: registri, etc., R1in, R1out, etc.), allora si ha bisogno di 4 bit per ogni fase.
- Se ho tre fasi si arriva ad una istruzione di 3*4 bit, 12 bit.
- Il decoder deve accettare istruzioni di N bit, se ho 2^N istruzioni diverse.
- Per ogni istruzione viene prodotta la decodifica come per esempio i 12 bit di cui sopra.
- Non e' detto che se ho 12 bit, ho 2^{12} istruzioni diverse, certe combinazioni sono impossibili o da evitare per evitare danni (per esempio aperture multiple di three state sullo stesso BUS).

Istruzioni con indirizzi

- ADD M[A1],R3,M[4F]
- La cui semantica risulta essere:
  - M[A1] = R3 + M[4F]
- Questo implica avere dentro la codifica dell'istruzione anche l'indirizzo (e.g., A1, 4F esadecimali)
- Se per ogni termine si puo' avere un indirizzo e lo spazio di indirizzamento e' di 12 bit, si hanno istruzioni con almeno 36 bit. 12x3

Data Path - 3 bus



L'istruzione

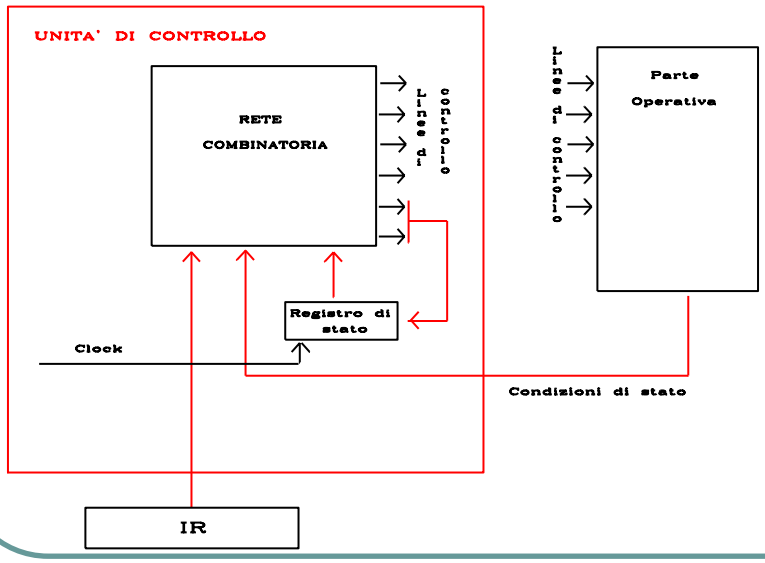
ADD R3,R2,R1

Cioe' R3=R2+R1

viene eseguita in un **solo clock**:

- R1_out; R2_out; ADD; R3_in

Logica cablata



Paolo Nesi, Univ. Firenze, Italy, 2003-07 287

Logica Cablata

- Tabella con ingressi:
  - IR +
  - registro di stato (incluso contatore delle fasi) +
  - variabili di stato relative alle uscite
- Contatore per le fasi (per esempio 4 bit controllare 16 registri/azioni)

Ist	cod1	cod2	cod3	STATO	OEi	STRi	C0	C1	C2	...
000	0010	0100	0000							
001										
010										

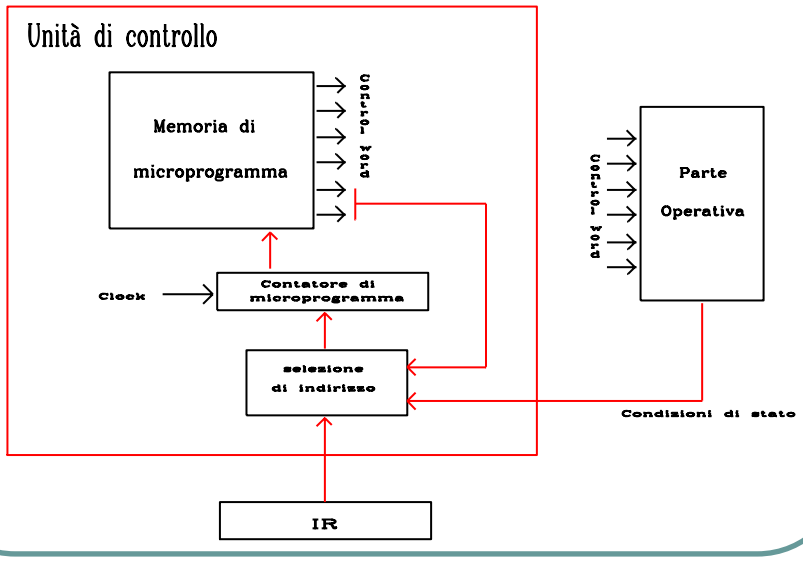
- F1 F2 F3

Paolo Nesi, Univ. Firenze, Italy, 2003-07 288

Logica cablata

- Progetto
  - Come sintesi di rete sequenziale
    - Sintesi per 1 (+di*), pochi 1, ROM sparsa
    - ingressi: IR, stato di UO
    - uscite: comandi (e.g., OE.., STR…, etc..)
  - Uso di ROM. La rete combinatoria ha come
    - ingressi (indirizzi alla ROM): IR, stato di UO, stato di UC (ALU flag register)
    - uscite: comandi che sono ingressi di eccitazione dei FF di stato, stato futuro
  - Logica programmabile (PLA)
  - Progettazione con CAD per VLSI
- Misura della complessità di UC:
  N_stati x N_ingressi x N_uscite

UC microprogrammata



UC microprogrammata

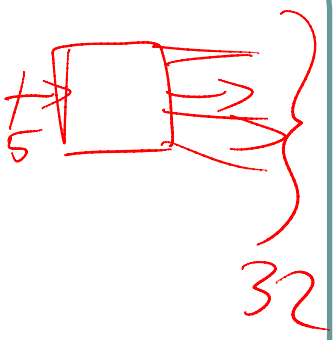
- Tecnica affermatasi negli anni 70
- UC è una sorta di calcolatore nel calcolatore
- La memoria di controllo contiene le microistruzioni:
- **mPC**: contatore di microprogramma. Contiene l'indirizzo della prossima microistruzione
  - All'inizio della fase di fetch mPC contiene l'indirizzo (I_0) del tratto di microprogramma corrispondente al fetch
  - Alla fine della fase di fetch mPC viene aggiornato con il contenuto (o ad una opportuna decodifica) di IR in modo da puntare alla microroutine che effettua le azioni richieste dalla particolare istruzione
  - Al termine, mPC viene di nuovo caricato con (mI0)

UC microprogrammata

- Differenti soluzioni:
  - sequenza di micro *control word* (CW)
  - microprogramma con subroutine
  - microistruzioni che contengono codificato al loro interno l'indirizzo della prossima microistruzione
  - *nanoprogrammazione:* le microistruzioni sono a loro volta interpretate da una unità nanoprogrammata (68000)
- Microprogrammazione orizzontale:
  - ogni bit di una CW corrisponde ad una linea di comando
- Microprogrammazione verticale
  - le CW contengono i comandi in forma convenientemente codificata

Esempio, ADD R3,R2,R1, microist..

1a IST	R1out	00001
	TEMPIin	00101
2a IST	R2out	01001
	TEMPIout	……..
	ADD	……..
	TEMPOin	……..
3a IST	TEMPOout	……..
	R3in	……..



N Bit di codifica della mIST, 2^N possibili mIST,
Buffer da controllare

Cablata o microprogrammata?

- Fino a fine anni '60: logica cablata (PDP8, HP 2116)
- Anni '70: microprogrammazione (VAX, Z80, 8086, 68000)
  - Repertorio di istruzioni molto esteso e variato: **CISC (complex instruction set computer)**
  - Il VAX 11/789 (Digital) e il 370/168 (IBM) avevano oltre 400.000 bit di memoria di controllo
- Dagli anni '80 si è tornati alla logica cablata;
  - Affermazione delle macchine **RISC (reduced instruction set computer)**
- Istruttivo è esaminare l'evoluzione dell'architettura Intel: da CISC a (praticamente) RISC

Ragioni per le CISC

- Un repertorio di istruzioni esteso è preferibile perché:
  - Istruzioni potenti semplificano la programmazione
  - Riduce il gap tra linguaggio di macchina e linguaggio di alto livello
  - *Software crisis*
- L'uso efficiente della memoria (all'epoca era costosa) era la preoccupazione principale:
  - meglio avere codici compatti
- Essendo (allora) la memoria di controllo molto più veloce della memoria centrale, portare funzionalità nella prima avrebbe migliorato le prestazioni della macchina

...Tuttavia

- Memorie RAM: molto più veloci delle precedenti a nuclei
- *Cache:* riducono ulteriormente i tempi di esecuzione
- Comportamento dei programmi:
  - l'80% delle istruzioni eseguite corrispondeva al solo 20% del repertorio.

  ⇒ *conviene investire nella riduzione dei tempi di esecuzione di quel 20%, anziché aggiungere raffinate istruzioni, quasi mai usate, ma responsabili dell'allungamento del tempo di ciclo di macchina*

  ⇒ *conviene costruire processori molto veloci, necessariamente con repertori semplici, e contare sull'ottimizzazione del compilatore*

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 5, L'architettura Software e la Scelta

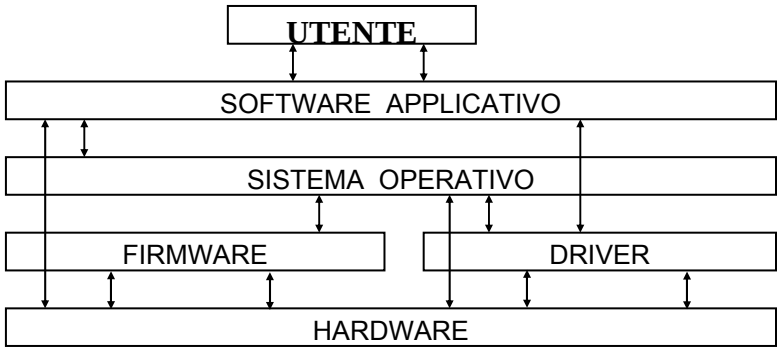
Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2007

Sommario della parte 5

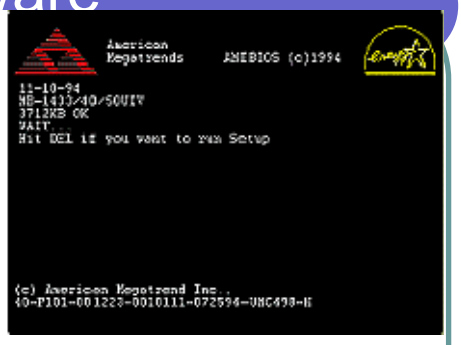
- **Hardware e Software**
- **L'architettura Software**
  - il sistema operativo
  - Il firmware, il BIOS
  - I driver
  - Il Boot
  - i programmi applicativi
- **Le Misure di un Elaboratore**
- **Le memorie di Massa**
- **Le prestazioni dei sistemi a microprocessore**

Hardware e Software

L'elaboratore è composto dall'hardware, i dispositivi fisici che lo costituiscono, e dal software, quelle procedure e istruzioni che ne dirigono le operazioni.



Hardware e Software



Il software è parte integrante del calcolatore poiché ne permette una maggiore o minore adattabilità alle richieste dell'utente stesso.
La struttura software è costituita da diverse categorie di programmi:

- B.I.O.S. (basic input output system)
- O.S.(operating system): KERNEL, UTILITY
- **SOFTWARE APPLICATIVO.**

Hardware e Software

Il BIOS

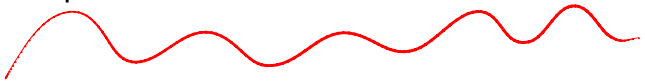
•Una particolare categoria di software è il BIOS (o firmware), un insieme di programmi, inseriti nell'hardware alla costruzione e copiati in memoria centrale all'avviamento (bootstrap).

•utilizzati dal sistema operativo per accedere alle risorse hardware del sistema. Grazie a questi programmi il sistema operativo può non considerare la struttura interna dell'elaboratore.

•All'acquisto di un personal computer si ha l'hardware ed il BIOS su cui si può implementare il sistema operativo più adatto a gestire, col software applicativo, le applicazioni più varie.

Hardware e Software

- Dal punto di vista logico funzionale l'utente interagisce con l'applicazione (schema precedente)
- Dal punto di vista fisico l'utente interagisce con l'Hardware
  - Quando viene effettuato un movimento del mouse questo produce l'attivazione di parti del codice del suo Driver, che a sua volta produce l'esecuzione di parti di istruzioni del SO e che a sua volta puo' produrre impatti sull'applicazione, come per esempio la tracciatura di una linea



Hardware e Software

- L'architettura a strati permette di astrarre la realizzazione ed il funzionamento delle applicazioni dall'HW, effettuando l'utilizzo dell'HW solo tramite chiamate tramite il SO
- Un'applicazione puo' accedere direttamente anche ai driver e all'hardware perdendo la caratteristica di essere indipendente dall'HW

Hardware e Software

IL SISTEMA OPERATIVO (OPERATING SYSTEM) è un insieme di programmi, che agisce da intermediario tra il calcolatore e l'utente, cosicché questi non debba interagire direttamente con l'hardware.
Il Sistema Operativo, rende possibile l'esecuzione del software applicativo in modo trasparente all'utente. L'utente consegna i dati prodotti e richiesti al sistema operativo, che accede all'hardware, senza che l'utente ne conosca i complicati meccanismi.

Esempi di sistemi operativi sono:
•MS DOS, WINDOWS 9x/ME/NT/2000/XP
•LINUX, UNIX VMX
•IBM OS2, MAC OS, ecc.

Le misure di un Elaboratore

- **Architettura:**
  - struttura della CPU: VN, HW, RISC, CISC, DSP
  - #Bit di CPU: tipico 32 o 64 bit
  - #bit di BA: tipico 16-32 bit
  - #bit di BD: tipico 32-64 bit
  - Clock consentito
- **Scheda Madre:**
  - per CPU singola o multipla, single core o a core multiplo
  - come sfrutta la CPU, il CHIP set: DMA, PIC, Bus Control, etc.
  - periferiche integrate: Disk Controller, raid, audio, USB, etc.
  - Memoria cache (min e max): tipico 512 Kbyte
  - Memoria RAM (min e max): tipico 512 Mbyte, 2, 8 Gbyte
  - Connessioni varie: rete, USB, seriale parallela, etc…..
  - Clock massimo e minimo, vari GHz
  - Tipo o tipi di BUS e loro velocità (clock):
    - AGP, PCI, EISA, VESA, etc. tipico 100 Mbyte
    - Numero di slot liberi

Elaboratore e le periferiche

- **Memoria di Massa**:
  - HD tipico 150 Gbyte, Raid (e.g., 0,1,2,3,4,5), etc.
  - FD, ZIP, Mcard, etc. (passate in disuso)
  - CD, DVD+R, DVD+-RW, etc.
  - Tapes (ancora in uso)
  - USB Memories: da diversi Mbyte ad alcuni Gbyte
- **User Interface:**
  - video, audio, joistick, mouse, game, etc.
- **I/O:**
  - USB, parallela, seriale, IRDA, 1394, etc.
- **Comunicazione:**
  - Rete fissa o wireless, velocità: 10-100 Mbps, 1Gbyte
  - Protocolli di comunicazione: TCP/IP, PPP, etc.
  - Modem: ADSL, GPRS, UMTS, etc.
  - etc.

Memoria di Massa:

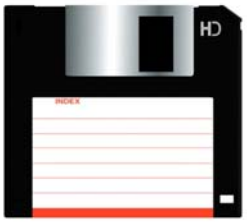
- **Mobili**
  - FD, Floppy disk (ormai non piu' in uso)
  - ZIP (ormai non piu' in uso)
  - CD, Compact Disk
  - DVD: DVD+R, DVD+-RW, etc.
  - Tapes
  - Memory card, etc. (ormai non piu' in uso)
  - USB memory from some Mbyte to few Gbyte
- **Fisse**
  - HD tipico 80-250 Gbyte,
  - Raid1 (solo duplicazione), Raid5, etc.
- Etc..

Memorie di massa - Floppy Disk

•Sono di materiale plastico ricoperti di materiale magnetico.

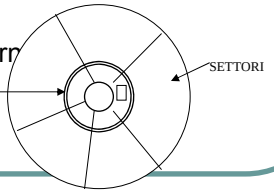
•Ognuno dei due lati è diviso in settori, divisi a loro volta in tracce. I settori sono degli spicchi mentre le tracce sono concentriche. I dischi magnetici sono letti per tracce concentriche. Capacità di 740K-1,44 MB.

Esiste nella parte centrale un riferimento, un foro, che permette l'identificazione della traccia 0.



Il tempo di accesso ad una traccia è in funzione della sua posizione sulla superficie del dischetto ed è mediamente di circa 25 millisecondi (l'accesso all memoria RAM è di circa 30 nanosecondi).

Si ha un accesso diretto (quando questa passa sotto la testina) alla traccia e sequenziale all'interno della traccia. Una traccia tipicamente contiene 512 o 1024byte.



Floppy da 8 e 5 e 1/4 pollici

160Kbyte – 1.2Mbyte

Floppy da 8 pollici

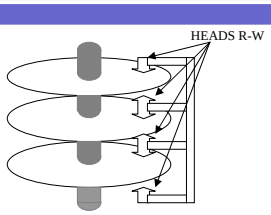
Floppy da 8, 5 e1/4, 3 e ½ Pollici

Paolo Nesi, Univ. Firenze, Italy, 2003-07 312

Memorie di massa - Hard Disk

Composto da dischi di materiale metallico, rigidi, generalmente d'alluminio, raccolti in una pila e sigillati all'interno di un contenitore dotato di un dispositivo di rotazione e di testine di lettura/scrittura.

Nell'hard disk, per trovare un'informazione è necessario conoscere la traccia, il settore, il lato/side e il disco sulla quale essa si trova.

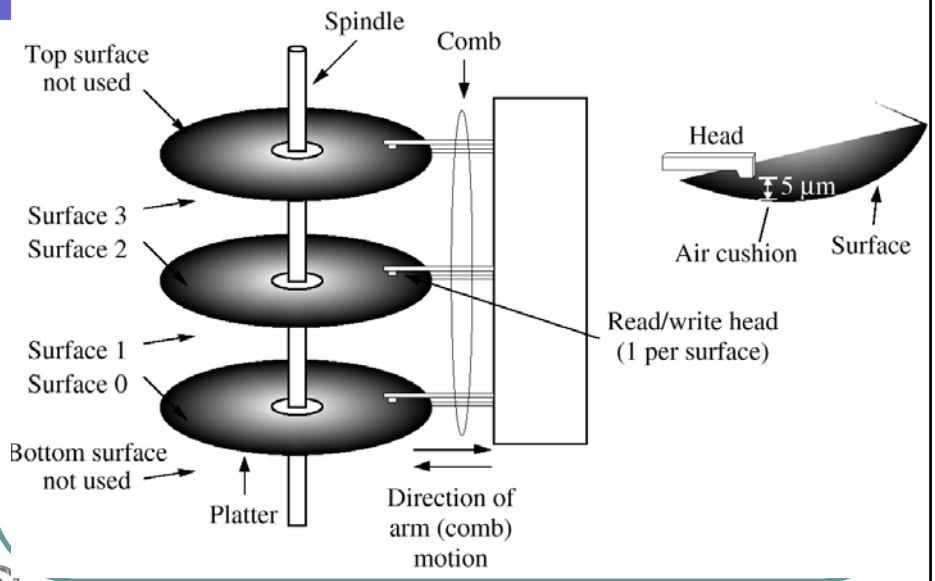


Hard Disk Maxtor Diamondmax Plus 9, **80gb Eide**, **Ultra Ata/133**, **8.7ms** Tempo Acc, **2mb Cache**, **7200 Rpm**

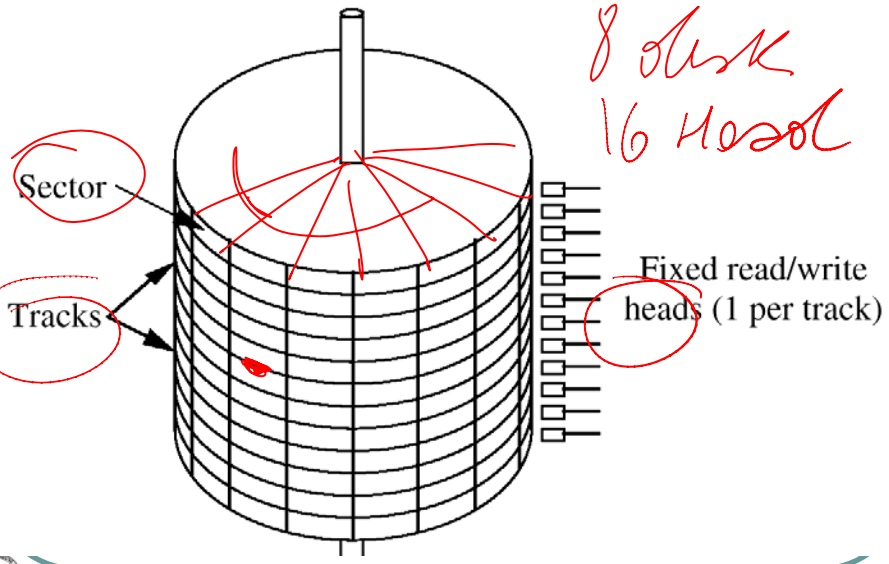
Poiché i dischi sono di materiale rigido, essi sopportano velocità più alte (RPM) ed il tempo di accesso per leggere un'informazione sull'hard disk è inferiore ai 10 millisecondi. La tecnologia impiegata è magnetica, le capacità raggiungibili fino a ~120GB.

Paolo Nesi, Univ. Firenze, Italy, 2003-07 313

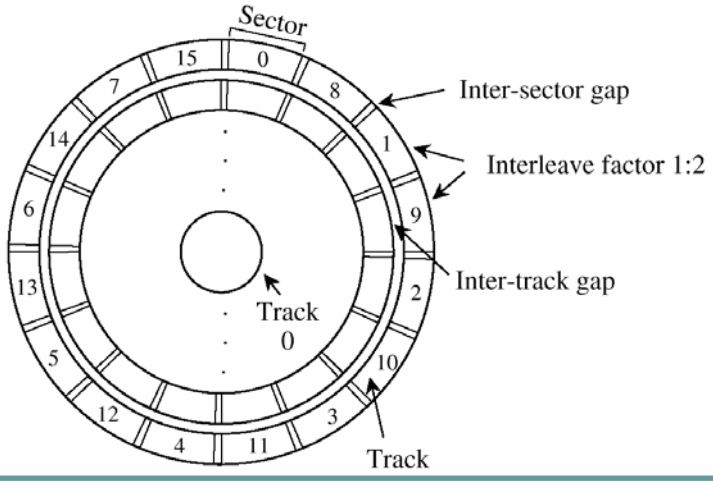
Hard Disk



Dischi Magnetici



Organizzazione dei dati nei dischi



Paolo Nesi, Univ. Firenze, Italy, 2003-07 316

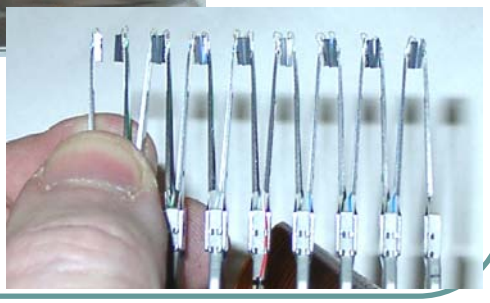
HD da 0,5 Mbyte, fine anni '70

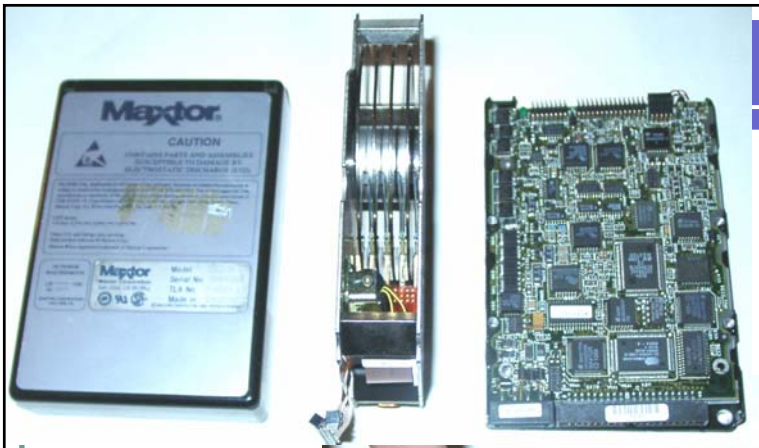
Paolo Nesi, Univ. Firenze, Italy, 2003-07 317

HD da 0,5 Mbyte, fine anni ’70, 35-40 Cm

HD da 5”1/4 con 4 side/head, 10Mbyte, fine anni ‘80

HD da 5”e1/4 con 16 side/head (400Mbyte, anni ‘90)





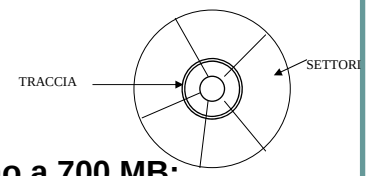
HD 3”1/2

8 side/head

Anni ’90-’00

Data CD (CD-Rom, CD-RW, DVD)

Il CD si legge per piste concentriche come il floppy, andando dall'interno verso l'esterno del disco. Il tempo di acceso è in funzione della velocità del motore in dotazione ma è dell'ordine di 1/(2-10) volte meno quello di un Hard Disk. La velocità di lettura si misura con multipli della velocità di lettura dei CD Audio (170 kbytes/sec = 1x) . Oggi sono in commercio modelli in grado di leggere anche 60x.



- Tecnologia impiegata: ***ottica***
- Capacità di memorizzazione:**CD, fino a 700 MB; DVD, fino a ~9GB, BluRay circa 50Gbyte**
- Velocità di accesso ai dati: ***media**, nell'ordine delle centinaia di ms*

CD and Floppy vari

Unità di Backup o a nastro

Le memorie a nastro sono supporti magnetici solitamente utilizzati per copie di riserva (backup). Il nastro non è una memoria ad accesso diretto ma ad accesso sequenziale: devo cioè svolgerlo ed avvolgerlo fino a trovare il dato che mi interessa (quindi il tempo di accesso può essere elevatissimo). La capacità dei nastri va da 60 Mbyte a 120 Gbyte circa.

I loro punti di forza stanno nella capacità di memorizzazione e nel basso costo.

Unità a nastro (DAT)

Tape Library

Data Cartridge vecchio tipo

Memorie di massa - Altri supporti removibili

IOMEGA ZIP

- Tecnologia impiegata: ***magnetica***
- Capacità di memorizzazione: ***100Mb***
- Velocità di accesso ai dati: ***media***

USB Memories ……

- Tecnologia impiegata: **flash memory**
- Capacità di memorizzazione: ***4Gbyte***
- Velocità di accesso ai dati: ***medio alta***

Interfacce utente

- **Output**
  - **Adattatori Grafici**
  - **Visori 2D, 3D**
  - **Stampanti, Printer**
  - **Etc.**
- **Input**
  - **Acq. Video**
  - **Mouse, 3D Mouse**
  - **Joystick**
  - **Tavoletta, tablet**
  - **Hand tracking**
  - **Touch screen**

I/O, Input/Output, Comunicazioni

- **Comunicazioni 1:1**
  - USB,
  - parallela,
  - Seriale,
  - IRDA,
  - 1394,
  - etc.
- **Comunicazioni N:M**
  - Rete fissa o wireless, velocità: 10-100 Mbps
  - Protocolli di comunicazione: TCP/IP, PPP, etc.
  - Modem: ADSL, GPRS, etc.
  - etc.

Le prestazioni

- MIPS (Milioni di istruzioni al secondo)

$$MIPS = N\_ist / (T\_CPU * 10^6)$$

E' un indice che ha poco significato

- MFLOPS (Milioni di istruzioni in virgola mobile al secondo)

$$MFLOPS = N\_vm / (T\_CPU * 10^6)$$

- Ci sono indici migliori
  - SpecINT 95
  - SpecFP 95

Il processore: prestazioni

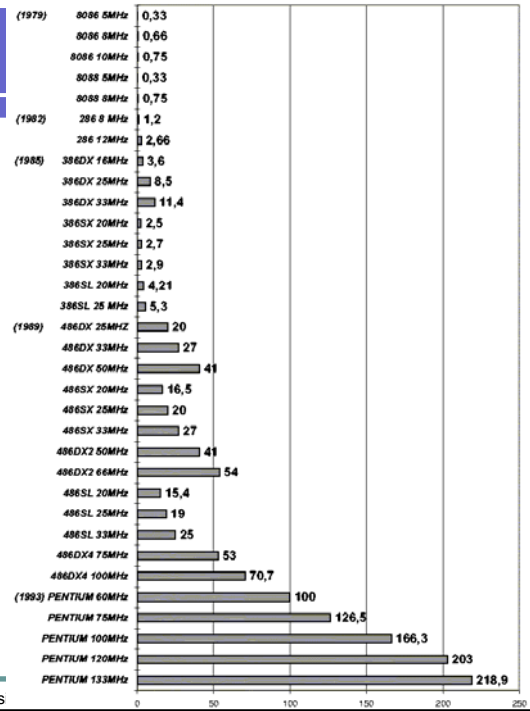
Esistono diversi criteri per valutare la *performance* di un processore.

Sono tutti più o meno discussi o discutibili.

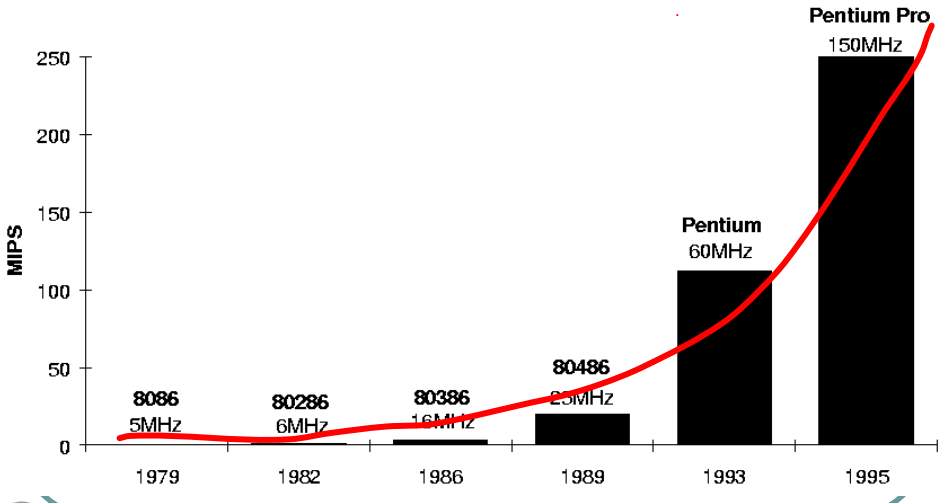
- Velocità di clock (GHz)
- Mips (millions instructions per second)
- MFlops (millions floating-point operations per second)
- iCOMP, iCOMP 2.0 (Intel COmparative Microprocessor Performance)
- P-rating, SPECint, SPECfp (real-world benchmarks)

MIPS

Da 8086 a Pentium Pro

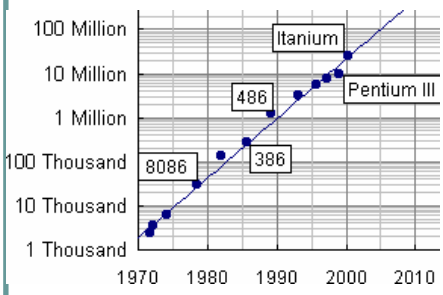


Da 8086 a Pentium Pro

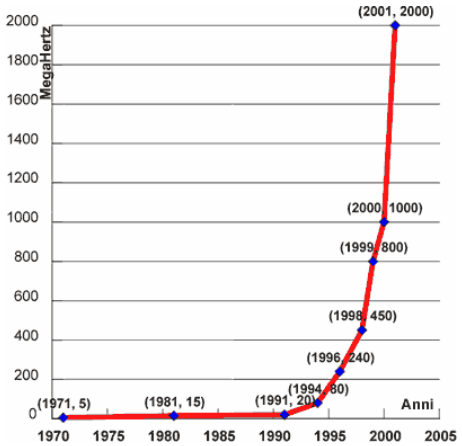


Evoluzione dei Microprocessori

Numero di transistor impiegati nei processori



La nuova frontiere e' l'integrazione di piu' CPU core in un'univa CPU fisica



Frequenza di clock dei processori

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 6, La CPU 8086

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
Gennaio 2006

L'8086, il data sheet

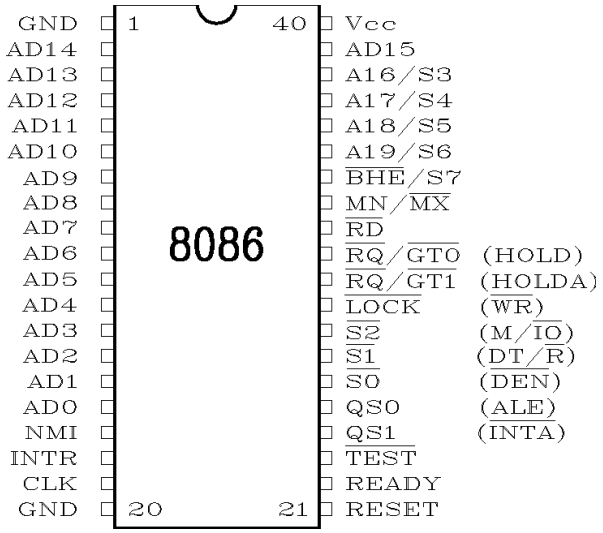
Features

- Compatible with NMOS 8086
- Completely Static CMOS Design
  - DC 5MHz (80C86)
  - DC 8MHz (80C86-2)
- Low Power Operation
  - ICCSB ..... 500μA Max
  - ICCOP ..... 10mA/MHz Typ
- 1MByte of Direct Memory Addressing Capability
- 24 Operand Addressing Modes
- Bit, Byte, Word and Block Move Operations
- 8-Bit and 16-Bit Signed/Unsigned Arithmetic
  - Binary, or Decimal
  - Multiply and Divide
- Wide Operating Temperature Range
  - C80C86 ..... 0°C to +70°C
  - I80C86 ..... -40°C to +85°C
  - M80C86 ..... -55°C to +125°C

Si consiglia di fare il "download" dal sito WEB del corso, si veda prima pagina, dei DATA SHEET (dei manuali tecnici operativi) dei componenti.

Tali documenti sono stampabili come il resto delle dispense e dei lucidi

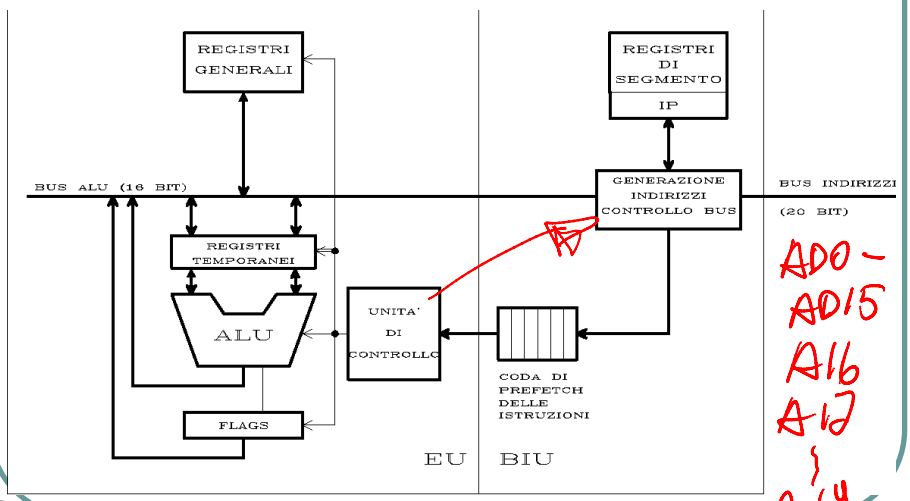
Piedinatura dell'8086



Caratteristiche

- 20 bit BUS address, 20 bit di indirizzamento
  - 1 Mbyte, da 00000H a FFFFFH
- 16 bit BUS dati, 16 bit per il BUS dati
  - Dati e istruzioni, architettura VN
- 16 bit ALU, operazioni algebrico e logiche native a 16
  - Complemento a 2 per interi, virgola mobile, etc.
- BUS Multiplato nel tempo
- Memoria Segmentata
- BUS con gestione del Tri-State
- Due Modalità di funzionamento:
  - modo minimo e modo massimo
  - Gestione semplice ed avanzata del BUS

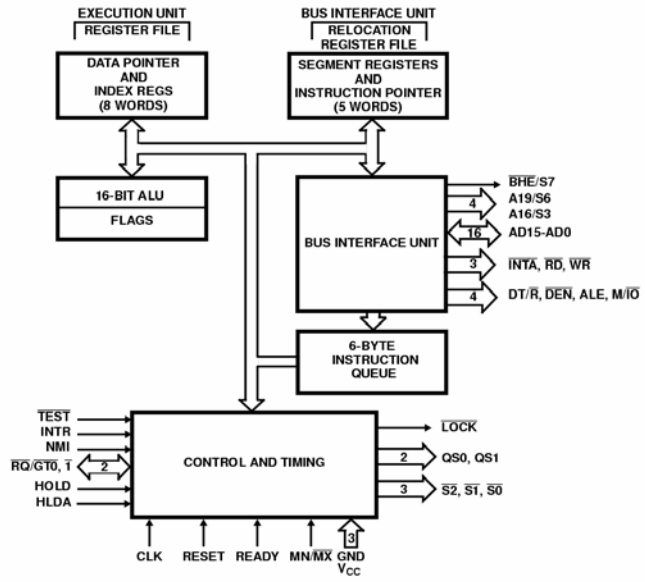
Architettura interna dell'8086



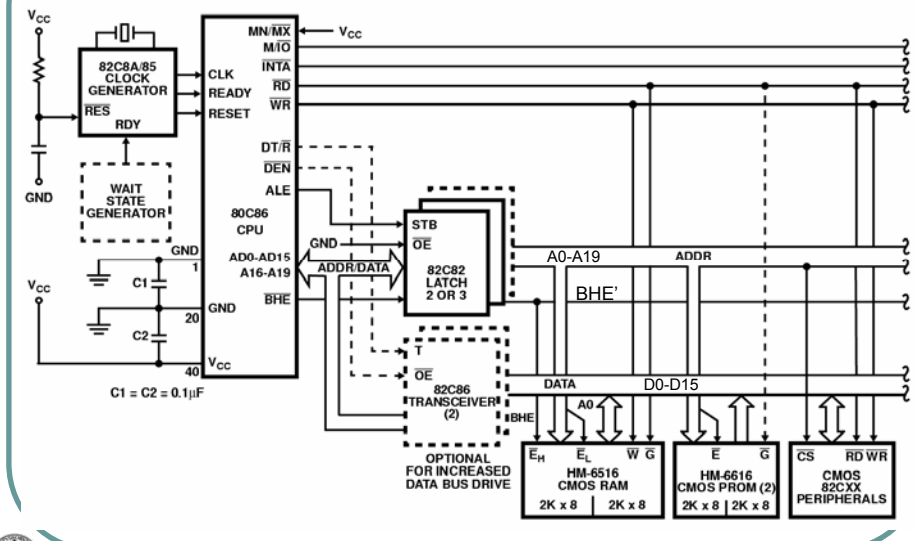
Componenti principali

- EC: Execution Unit
  - Esecuzione delle operazioni, ALU e Registri
- BIU: Bus Interface Unit
  - Accesso alla memoria per istruzioni e dati
  - Gestioni dei registri di indirizzamento
    - Calcolo dell'indirizzo fisico da quello logico di segmento
  - Controllo logico del BUS
  - Acquisizione delle istruzioni ed inserimento di queste in una coda

Diagramma Funzionale



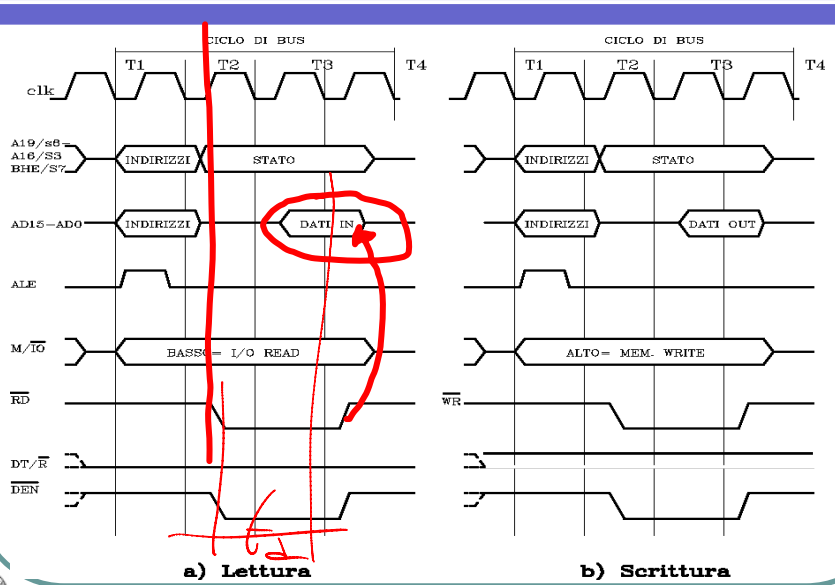
Modo Minimo dell'8086



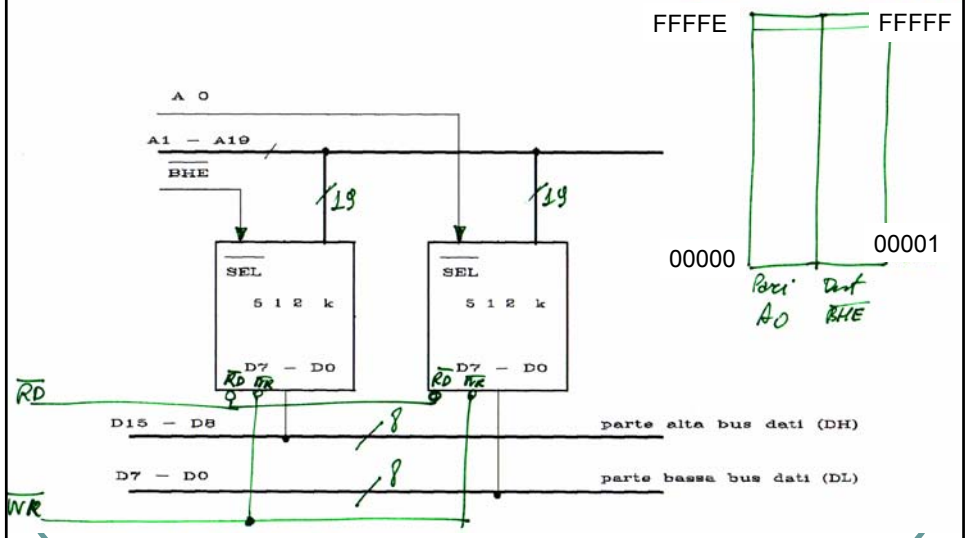
BUS Multiplato, modo minimo

- 16 bit vengono presentati nella prima parte del ciclo macchina sui pin ADxx
- Sugli stessi pin si ha il BUS dati nella seconda parte del ciclo macchina
- I bit di indirizzo da 17 a 20 sono presentati su pin diversi che nella prima parte del ciclo macchina presentano tali indirizzi, mentre nella seconda lo stato della CPU

Ciclo Macchina, Accesso alla Memoria



Accesso a BUS con BHE



Selezione/Combinazione delle Memorie

- Il Segnale di SEL negato sulle memorie, anche chiamato Chip Select negato, (CS), permette di effettuare delle composizioni di memorie.
- Realizzare dei banchi di memoria di dimensioni maggiori a partire da tagli piccoli
- Nell'esempio viene scelto il chip sulla base del bit meno significativo.
  - Questa soluzione permette di fare letture parallele dei due byte
- Se si fosse scelto di dividere la memoria in due blocchi sulla base del bit piu' significativo si avrebbe avuto i primi 512 Kbyte in un chip ed i secondi in un altro,
  - Questo non e' conveniente
  - in tale caso la lettura non sarebbe potuta essere di una Word a 16 ma solo di un byte alla volta

Tipo di Lettura sul BUS

- Not BHE, BUS High Enable, abilitazione della parte alta del bus, segnale attivo basso
- Letture di Byte e Word da indirizzi pari e dispari ?

not BHE	A0	dati	start	#cicli
1	0	byte	pari	1
0	1	byte	dispari	1
0	0	word	pari	1
1	1	--	--	--

Paolo Nesi, Univ. Firenze, Italy, 2003-07 348

Lettura di una Word da un dispari

					…	…	…	…	A3	A2	A1	A0
…	…	…	x	x	x	x	x	x	x	x	0	0
…	…	…	x	x	x	x	x	x	x	x	0	1
…	…	…	x	x	x	x	x	x	x	x	0	1
…	…	…	x	x	x	x	x	x	x	x	1	0

- Nel caso di Byte Consecutivi a partire da un indirizzo pari i bit sono diversi solo per A0
- Nel caso di Byte Consecutivi a partire da un indirizzo dispari i bit sono diversi anche per A1, non si puo' fare con una configurazione HW come quella precedente.

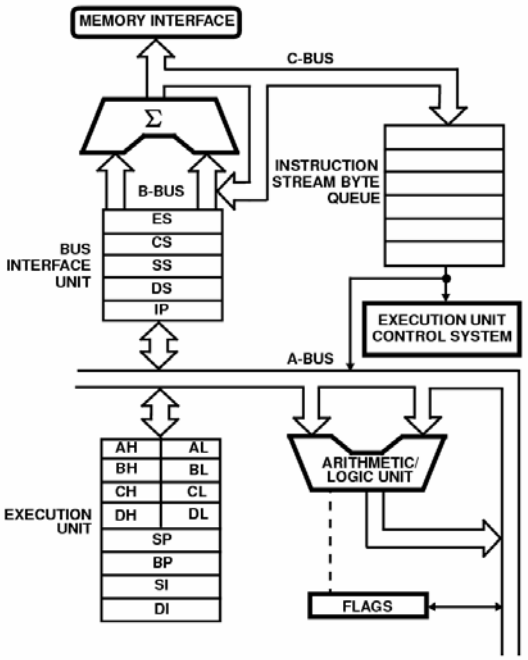
Paolo Nesi, Univ. Firenze, Italy, 2003-07 349

Registri Interni

Si noti il parallelismo interno fra la BIU e la EU.

Possibile perche'

- •Presenza delle coda
- •Registri diversi per gestione indirizzi e dati
- •ALU indipendente per calcolo indirizzo fisico



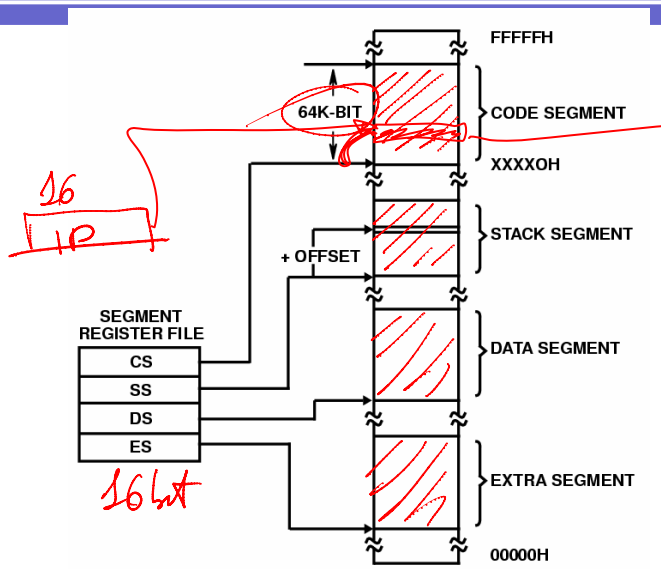
Paolo Nesi, Univ. Firenze, Italy, 2003-07 350

Registri della BIU,

- **IP: Instruction Pointer (offset)**
  - Non e' semplicemente l'offset dell'istruzione successiva poiche' ho la coda di istruzioni.
- **CS: Code Segment**
  - Segmento che contiene codice, il programma, corrente o meno
- **SS: Stack Segment**
  - Segmento che contiene dati temporanei. In tale segmento viene salvato lo stato della CPU e del programma quando si hanno dei cambi di contesto per ripristinarlo quando serve, se ne parlerà piu' in dettaglio in seguito
- **DS: Data Segment**
  - Segmento che contiene dati per l'elaborazione
- **ES: Extra Segment**
  - Segmento aggiuntivo per i dati
- Tutti questi registri sono a 16 bit mentre la memoria e' a 20 bit
- Servono per comporre il valore dell'indirizzo fisico
- Si hanno indirizzi logici a 16 bit e fisici a 20 bit.

Paolo Nesi, Univ. Firenze, Italy, 2003-07 351

Memoria Segmentata



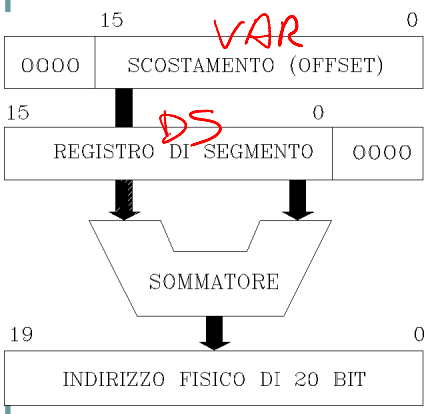
Paolo Nesi, Univ. Firenze, Italy, 2003-07 352

Memoria Segmentata

- La memoria e' di 1Mbyte, 20 bit
- Indirizzi logici sono espressi in termini di Segment:offset
- Segment e Offset sono parole di 16bit, 1 Word
- Questo permette di muovere i vari segmenti all'interno della memoria, fare rilocazione
- Anche di sovrapporre i segmenti
- L'indirizzo fisico e' ottenuto da quello logico:
  - Indirizzo fisico = segment * 10H + offset
- I 16 bit dei numeri sono memorizzati in memoria in ordine inverso.

Paolo Nesi, Univ. Firenze, Italy, 2003-07 353

Calcolo dell'indirizzo fisico

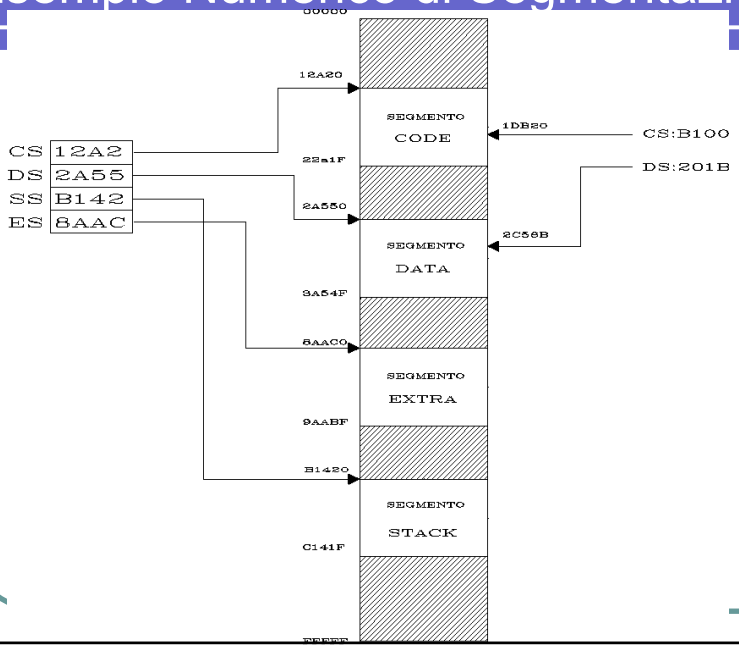


- ADD AX, VAR
  - AX ←AX+M[DS:offset(VAR)]
- ADD AX,ES:ALT
  - AX←AX+M[ES:ALT]
- Ogni programma deve dichiarare quali e quanti segmenti utilizza

Registro Accumulatore, AX

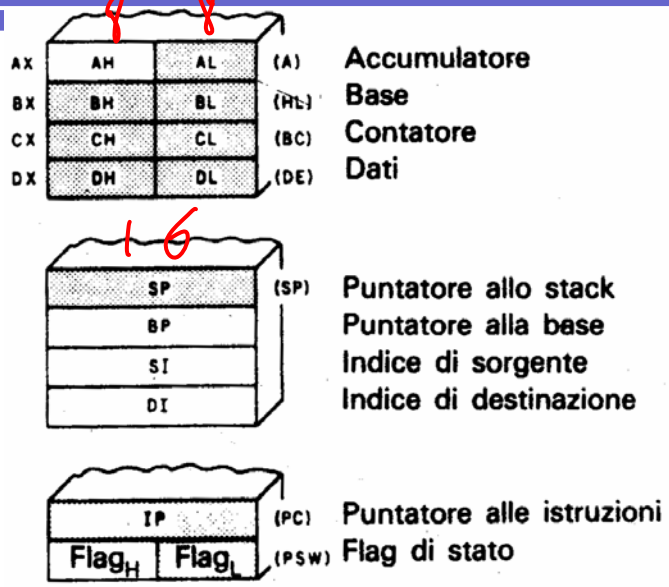
- Nell'architettura 8086 come in altre CPU Intel.
  - AX viene detto anche registro accumulatore
  - E' un registro privilegiato per certe istruzioni
- Per esempio ADD AX,VAR effettua una somma del valore i VAR e lo somma al valore di AX mettendo, accumulando il risultato in AX stesso:
  - AX ← AX + M[DS:VAR]

Esempio Numerico di Segmentazione



357

Registri di utilizzo generale



Paolo Nesi, Univ. Firenze, Italy, 2003-07

358

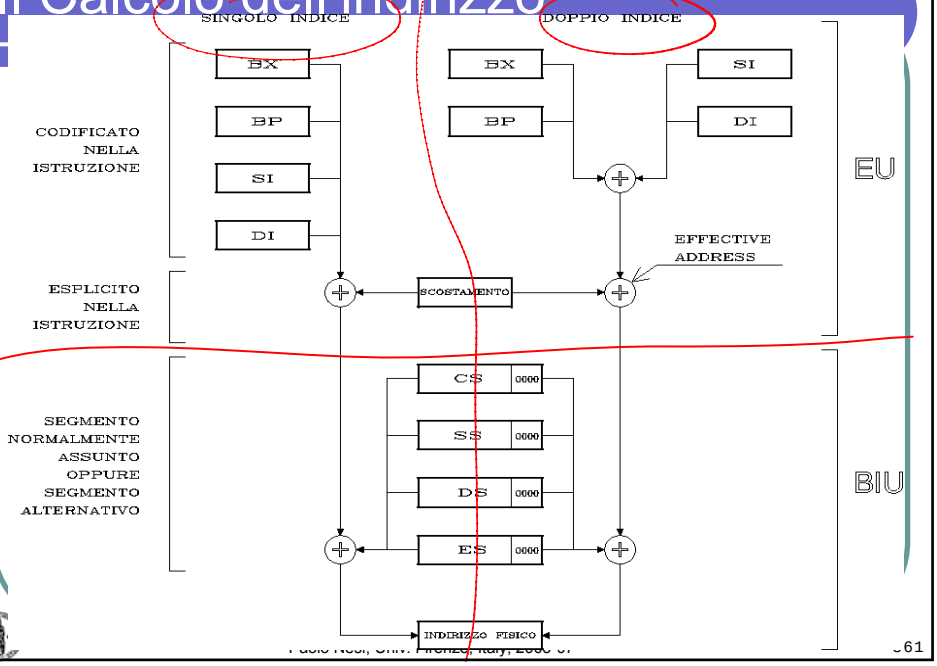
Registri "Specializzati"

- **AH-AL, BH-BL, CH-CL, DH-DL**
  - 8 registri a 8 bit che
  - possono essere visti anche come 4 registri a 16 bit: AX, BX, CX, DX
- **SP: Stack Pointer, 16 bit**
  - offset per indirizzare alla testa dello Stack dentro il SS
- **BP: Base Pointer, 16 bit**
  - offset per indirizzare all'interno dello Stack la sua posizione di riferimento
- **SI: Source Index, 16 bit**
  - Indice che serve per alcune istruzioni per indicare la posizione relativa della sorgente dei dati
- **DI: Destination Index, 16 bit**
  - Indice che serve per alcune istruzioni per indicare la posizione relativa di destinazione dei dati

Coinvolgimento dei Registri di indiriz.

TYPE OF MEMORY REFERENCE	DEFAULT SEGMENT BASE	ALTERNATE SEGMENT BASE	OFFSET
Instruction Fetch	CS	None	IP
Stack Operation	SS	None	SP
Variable (except following)	DS	CS, ES, SS	Effective Address
String Source	DS	CS, ES, SS	SI
String Destination	ES	None	DI
BP Used As Base Register	SS	CS, DS, ES	Effective Address

Il Calcolo dell'indirizzo



61

Esempi

- [CS: BX + SI + Var]
- [DS: BX + VAR]
- [SS: BP + FEH]

- La CPU non ha nessuna conoscenza del tipo di dato. I dati vengono interpretati dalle istruzioni per quello che sono: cioè numeri binari.
- Se l'istruzione fa l'ipotesi di lavorare su numeri complemento a 2 e i valori dati non lo sono (o viceversa), ovviamente la CPU non se ne puo' accorgere.

Paolo Nesi, Univ. Firenze, Italy, 2003-07

362

Registri Interni di stato

- **IP: Instruction Pointer, 16 bit**
  - E' sempre un Offset per un indirizzamento logico
- **PSW: program status word**,
  - 16 bit, non tutti utilizzati, solo flag singoli da un bit
  - Detto anche: Status Register o Flag register
  - Composto da una parte alta e bassa
  - I flag sono utilizzati dalle istruzioni per
    - tenere conto del contesto, del risultato delle istruzioni precedenti
    - modificare lo stato per le istruzioni successive, passare un riporto, condizionare una scelta, etc.

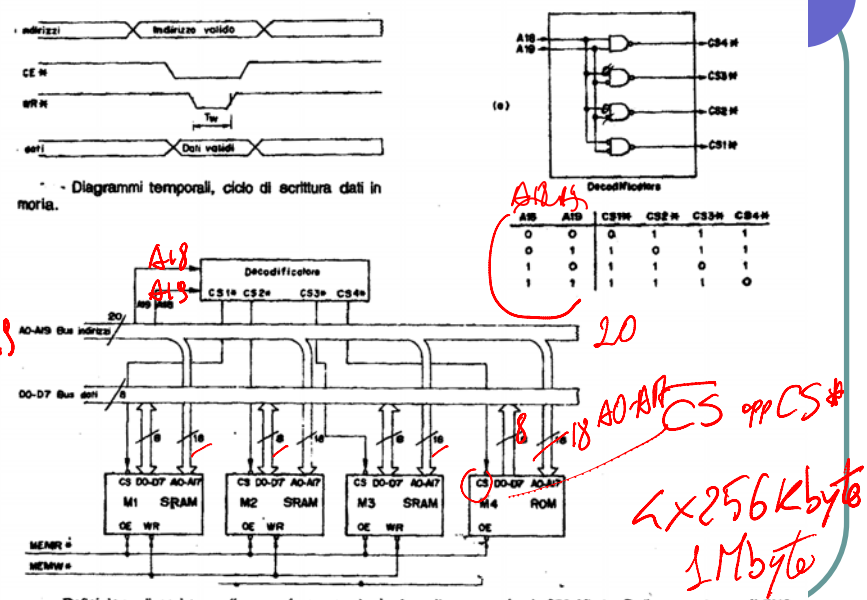
PSW, Flag Register

- **ZF (Zero Flag) -- Flag di zero (zero/nonzero).**
  - posto ad 1 quanto il risultato di una operazione aritmetica e' zero;
- **SF (Sign Flag) -- Flag di segno** (negativo/positivo).
  - posto a 1 quando il risultato di una operazione sulla ALU e' un numero negativo
- **CF (Carry Flag)** -- Flag di presenza di riporto (carry) (si/no)
  - posto ad 1 quando si ha un riporto nelle operazioni di somma o sottrazione sia a 8 che a 16 bit;
- **PF (Parity Flag)** -- Flag di presenza di parità.
  - posto ad 1 se la parte bassa (8 bit) del risultato di una operazione dell'ALU contiene un numero pari di 1;
- **AF (Auxiliary Flag)** -- Flag ausiliario (si/no)
  - posto ad 1 in presenza di riporto nelle operazioni in codifica BCD;

PSW, Flag Register

- **DF (Direction Flag)** -- Flag di direzione (decrescente/crescente),
  - assume significato solo nelle operazioni con stringhe dove identifica l'organizzazione di queste in memoria;
- **IF (Interrupt Flag) --** Flag che segnala le condizioni della CPU rispetto alle interruzioni (abilitate/disabilitate)
  - se e' ad 1 il programma in esecuzione puo' essere interrotto;
- **OF (Overflow Flag)** -- Flag di presenza di overflow (si/no)
  - il quale assume significato solo in presenza di operazioni aritmetiche
- **TF (Trap Flag)** -- Flag di presenza di trap (si/no),
  - quando e' ad 1 il programma può essere eseguito passo passo (single-step). utilizzato per il controllo dell'esecuzione del programma (debug).

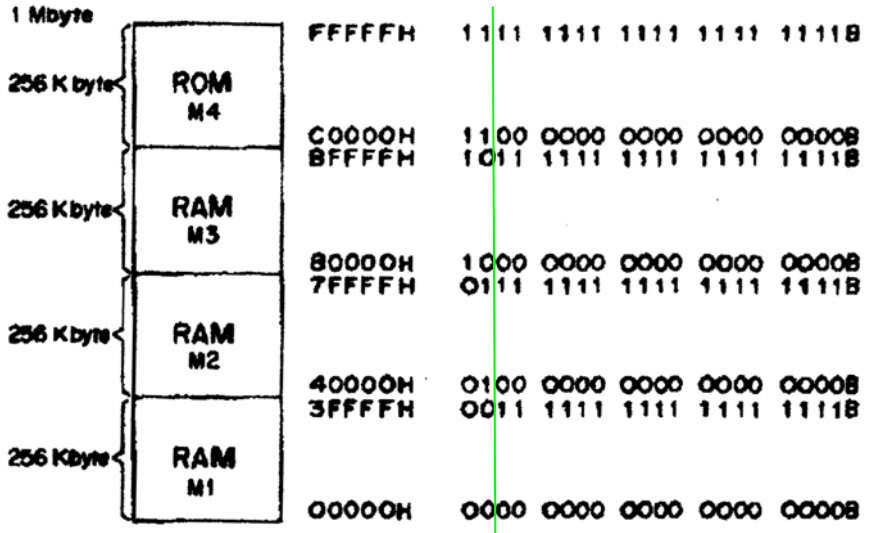
Esempio di Espansione della Memoria



- Diagrammi temporali, ciclo di scrittura dati in moria.

- Definizione di un banco di memoria composto da 4 quattro memorie da 256 Kbyte. Collegamento con il BUS stema.

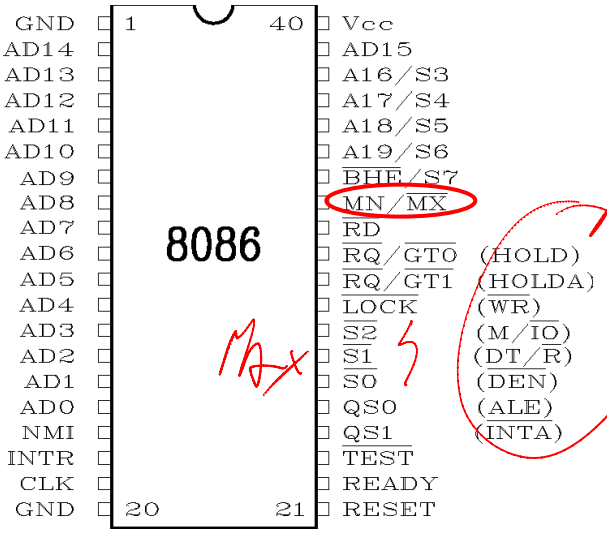
Mappa della memoria dell'esempio



Mappa della memoria.

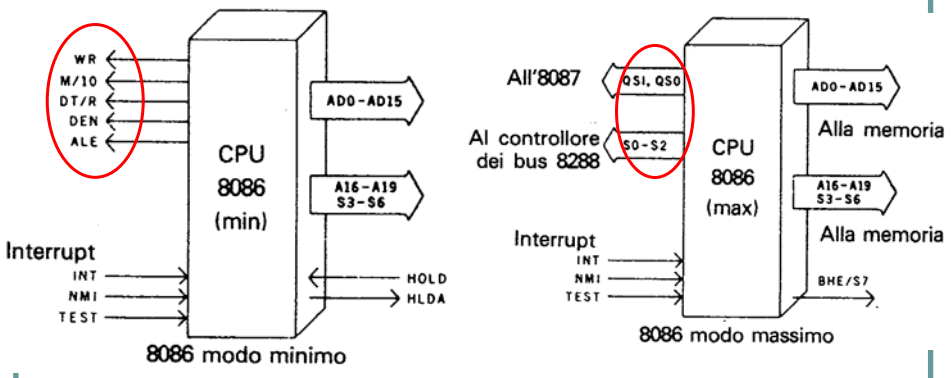
A18	A19	$\overline{CS1}$	$\overline{CS2}$	$\overline{CS3}$	$\overline{CS4}$
A18	A19	CS1*	CS2*	CS3*	CS4*
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Piedinatura dell'8086



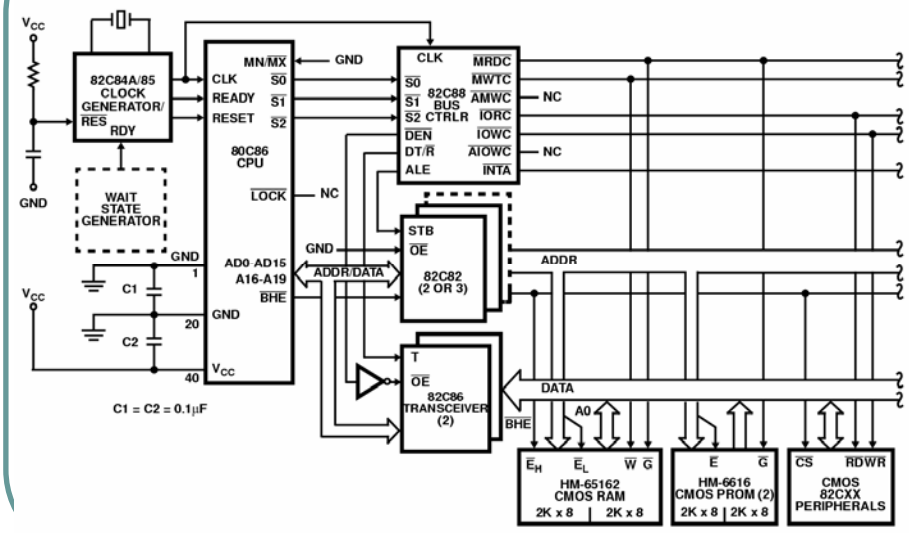
Paolo Nesi, Univ. Firenze, Italy, 2003-07 370

Modo Minimo e Massimo

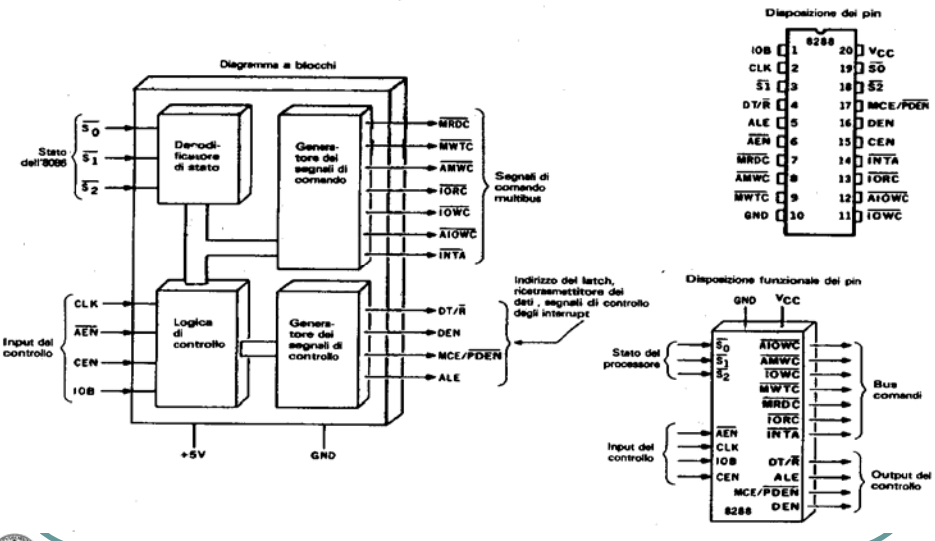


Paolo Nesi, Univ. Firenze, Italy, 2003-07 371

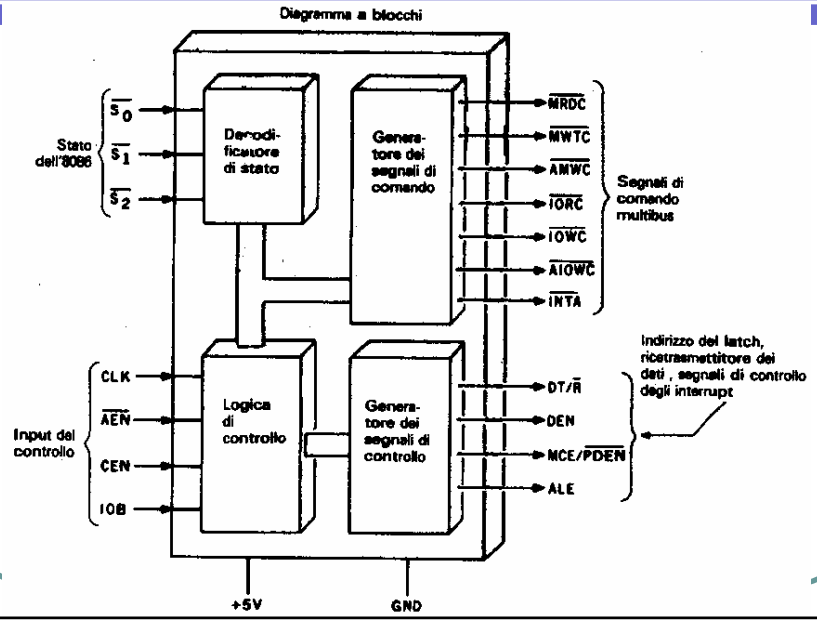
Modo Massimo dell'8086



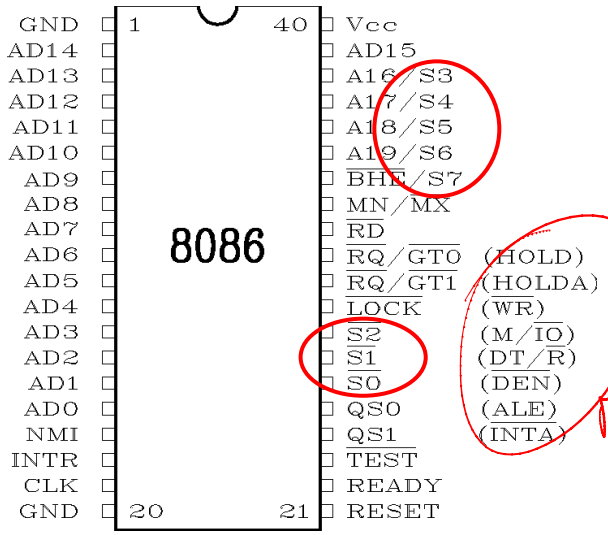
Il Controllore di BUS, 8288



Il Controllore di BUS, 8288



Piedinatura dell'8086



Paolo Nesi, Univ. Firenze, Italy, 2003-07 376

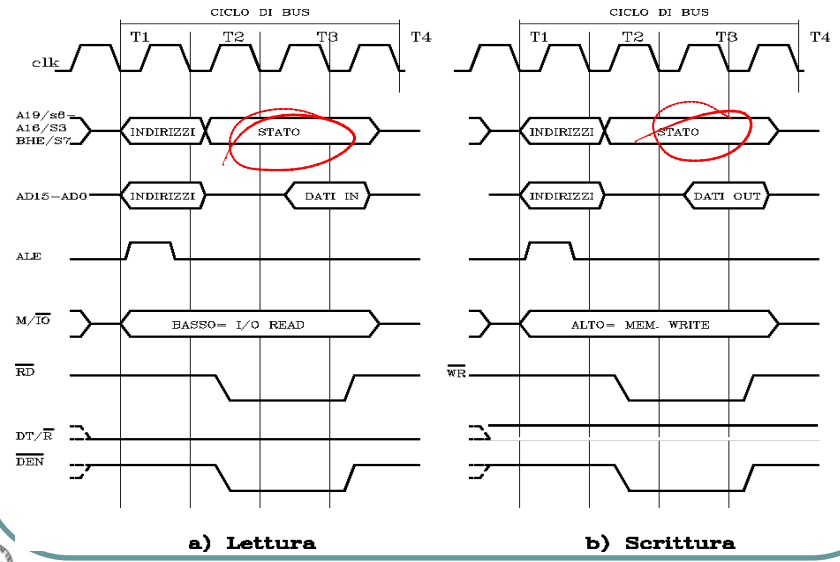
Lettura dei Bit di stato

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	CHARACTERISTICS
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

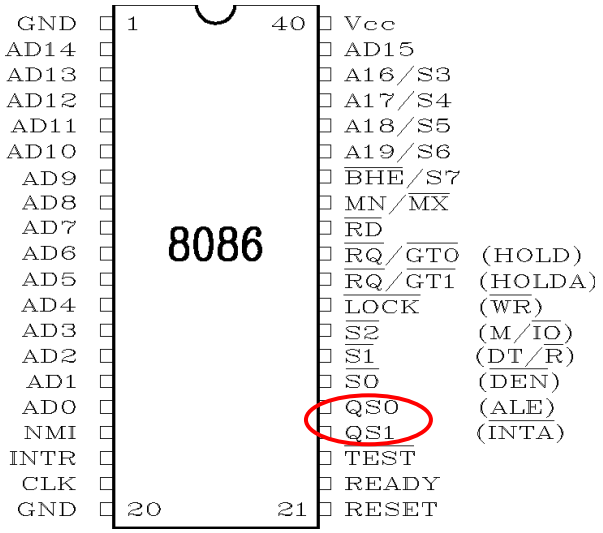
S4	S3	CHARACTERISTICS
0	0	Alternate Data
0	1	Stack
1	0	Code or None
1	1	Data

- Leggibili durante il ciclo macchina
- Letti da certe periferiche per la loro sincronizzazione
- Modo massimo
- Riconoscimento del tipo di ciclo macchina

Ciclo Macchina, Accesso alla Memoria



Piedinatura dell'8086

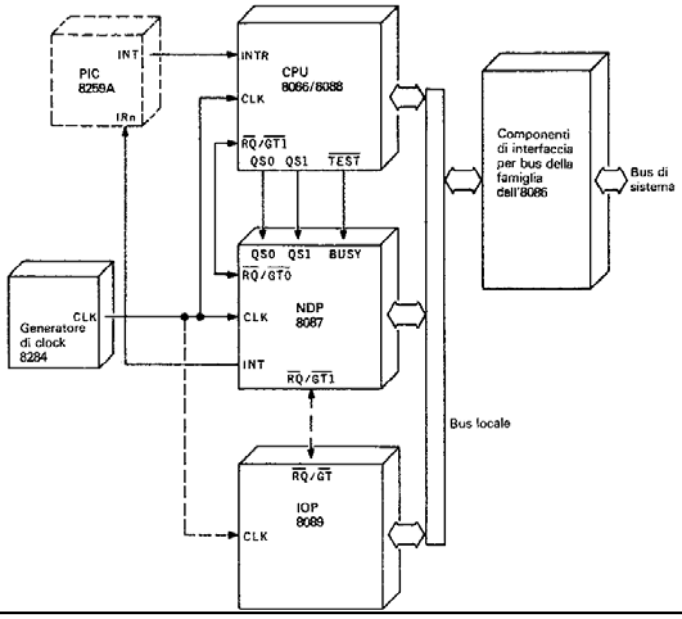


Lo stato della Coda

QSI	QSO	
0	0	No Operation
0	1	First byte of op code from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

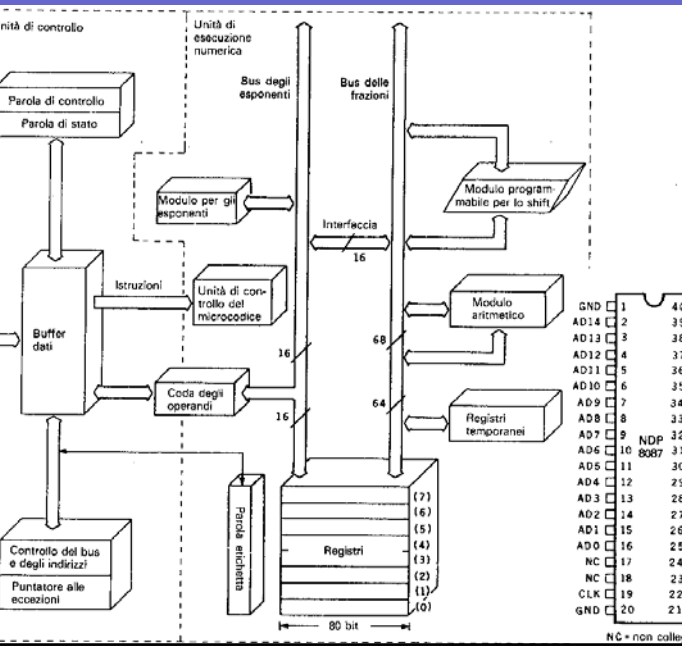
- Leggibili durante il ciclo macchina
- Letti da certe periferiche per la loro sincronizzazione
- Riconoscimento del tipo di ciclo macchina

8086 Massimo & 8087



381

Coprocessore Matematico 8087



382

Le Interruzioni

- Le interruzioni sono situazioni eccezionali che portano la CPU ad interrompere il normale svolgimento previsto del programma.
- La CPU salva **il contesto** e passa all'esecuzione di un altro spezzone di codice
- La CPU lavora sempre: cursore, movimento del mouse, icone animate, acceso al disco, etc.
- Interrupt: tastiera, mouse, stampante, disco, timer, etc.

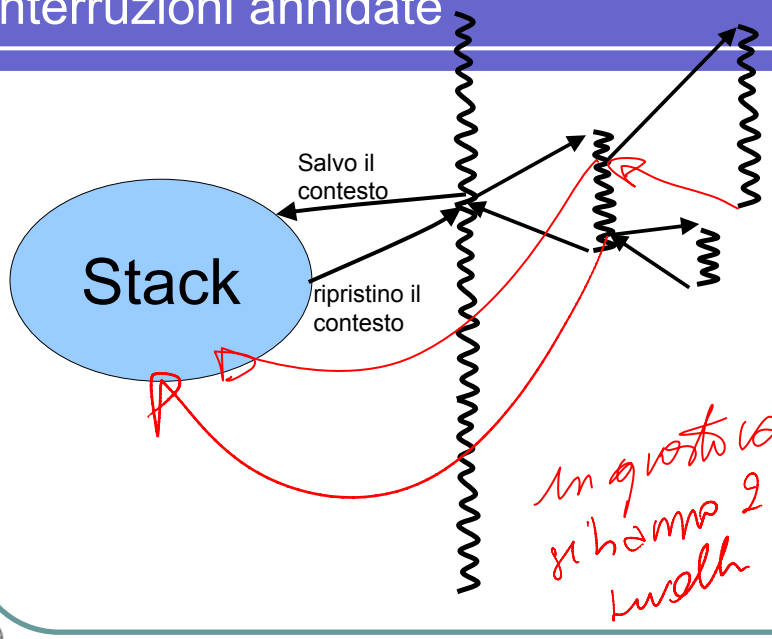
Paolo Nesi, Univ. Firenze, Italy, 2003-07 383

Le Interruzioni possono essere

- Le Interruzioni Hardware: se provocate da un periferica e quindi sono scaturite da un segnale fisico
- Software: codificate in chiaro nel codice per esempio INT 21
- trap o eccezioni: sono prodotte dall'interno della CPU da situazioni di errore (divide per zero, stack overflow, etc.) o di gestione ad elevata priorità

Paolo Nesi, Univ. Firenze, Italy, 2003-07 384

Interruzioni annidate



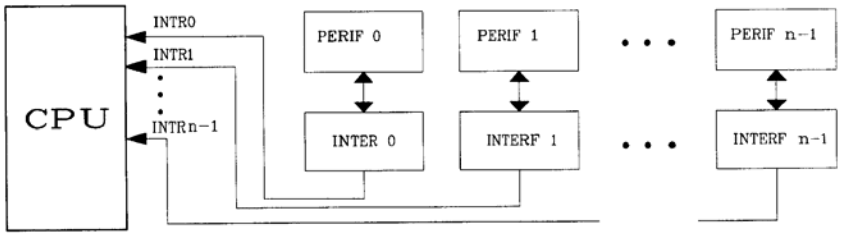
Le Interruzioni

- **La CPU deve prevedere l'arrivo di interruzioni predisponendo le routine/procedure che devono essere eseguite quando il contesto cambia.**
  - Al completamento della routine, l'esecuzione ritorna al programma che era stato interrotto
  - E' possibile interrompere anche le routine stesse con altre interruzioni. Questo può essere evitato con opportune istruzioni che abilitano e disabilitano la sensibilità alle interruzioni della CPU.
  - Il cambio di contesto scatena il salvataggio di questo nello Stack.
- **Le interruzioni possono essere disabilitate**
  - Sui pin della CPU di solito esiste anche un pin di Interruzione non disabilitabile, NMI, Non Maskerable Interrupt

Le Interruzioni

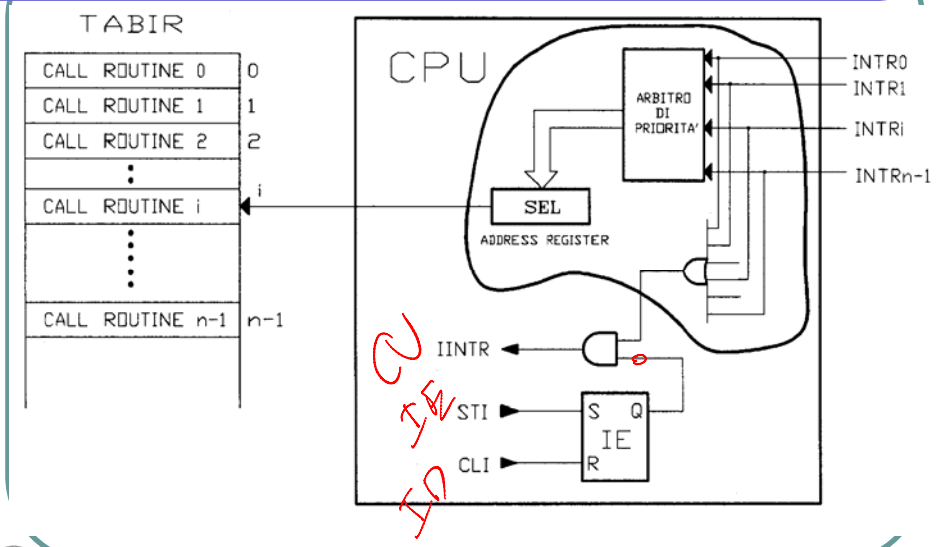
- **La CPU tipicamente può ricevere diverse interruzioni**
- Alcune interne altre da periferiche esterne, quelle esterne sono definibili dal progettista e anche in parte dall'utente
- alcune CPU hanno dei pin specifici per le interruzioni, tanti pin quante interruzioni possibili esterne (Interruzioni Indipendenti)
- Altre CPU hanno solo un pin e posseggono un meccanismo per riconoscere quale periferica delle tante ha scatenato tale interruzione. In questo caso, l'informazione che arriva alla CPU deve indicare quale procedura di servizio deve essere attivata.

Interruzioni a linee indipendenti

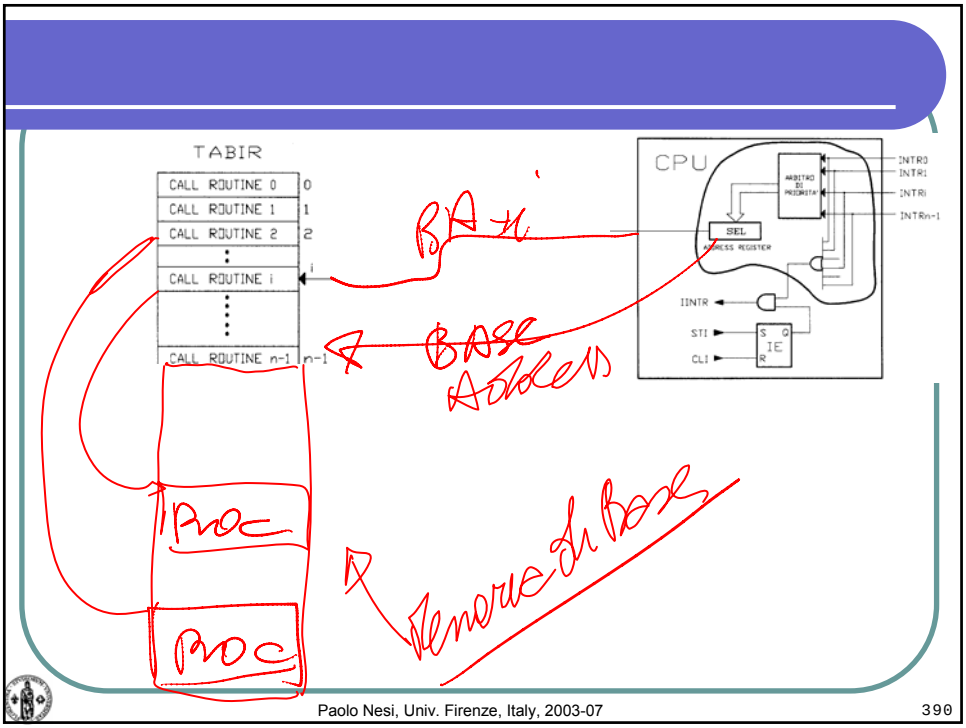


- Piuttosto complesso poiché la Cpu deve avere linee e pin indipendenti
- La priorità viene programmata internamente alla CPU o dalla posizione

Interruzioni a Linee indipendenti



Paolo Nesi, Univ. Firenze, Italy, 2003-07 389



Paolo Nesi, Univ. Firenze, Italy, 2003-07 390

Servizio delle Interruzioni

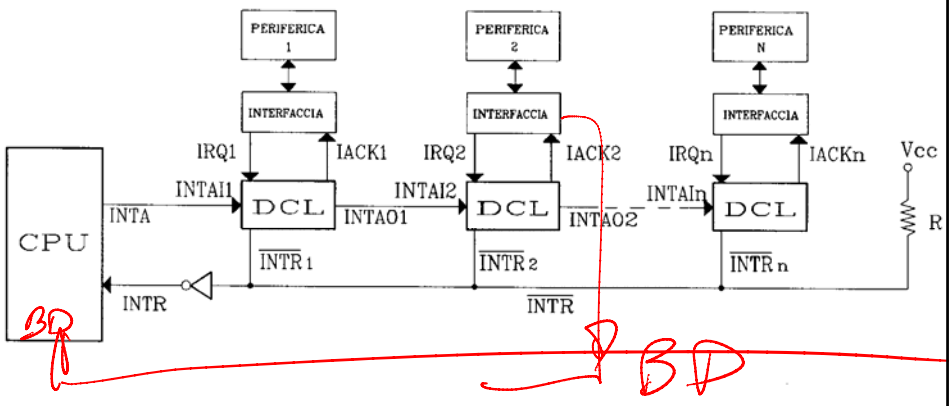
- E' necessario identificare la richiesta da servire, questa può essere diretta oppure decodificata se l'interruzione e' vettorizzata
- Nel caso in cui la CPU non riceva indicazioni dirette della periferica che ha effettuato la richiesta vi sono due metodi per arrivare ad identificarla:
  - Polling (la CPU richiede a tutte le periferiche in modo ciclico se sono state loro fino a che non identifica quella che ha provocato l'interruzione)
  - Daisy Chain (esiste una catena Hardware che risolve il problema)

Polling delle interruzioni

- Scansione di tutte le periferiche e colloquio con queste una ad una per capire quale di queste ha fatto richiesta
- La scansione (polling) puo' essere eseguita secondo un certo ordine. Tale ordine determina la priorità se la scansione si ferma alla prima periferica che afferma di avere effettuato richiesta.
- in tale caso, le altre periferiche perdono il servizio se hanno priorità piu' bassa, e devono richiederlo in seguito se vogliono essere servite.

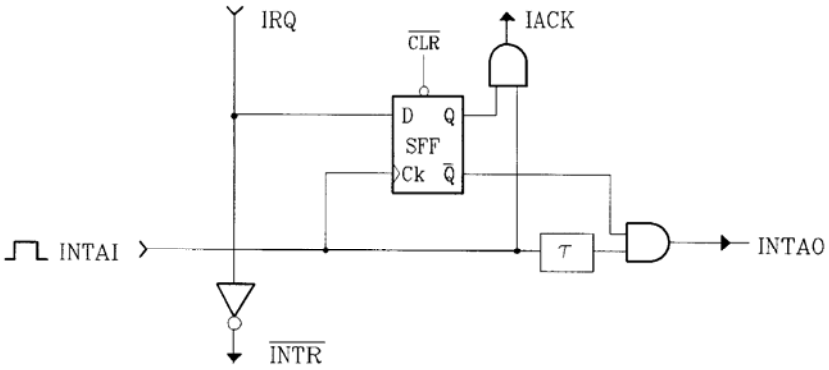
Daisy Chain Asincrona

- Si ha una catena di dispositivi la cui priorità è determinata dalla posizione



- INTR è dato dal *wired* OR di tutte le interfacce
- INTA viene propagato lungo la catena

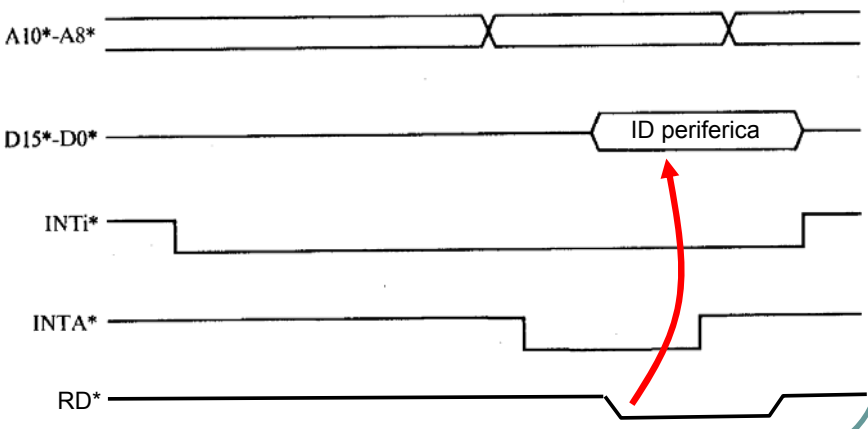
Interfaccia DCL, Daisy Chain Logic



- L'elemento di ritardo impedisce che INTA venga trasferito a valle prima che SFF abbia commutato

Un esempio di ciclo di INTA

- Non e' un 8088/8086



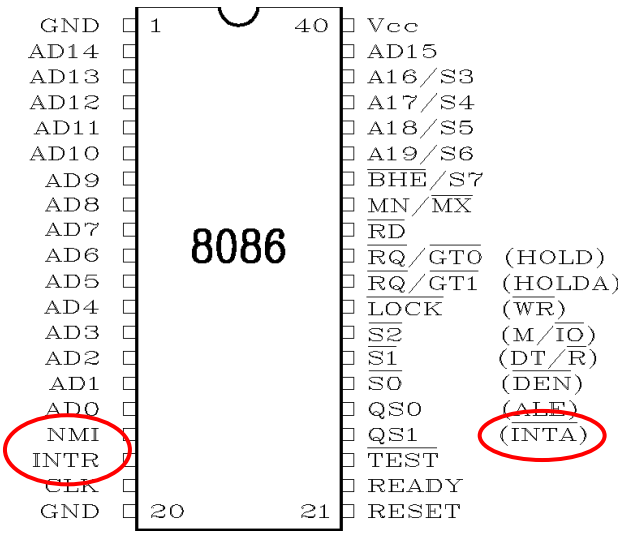
Paolo Nesi, Univ. Firenze, Italy, 2003-07 395

Le Interruzioni e l'8086

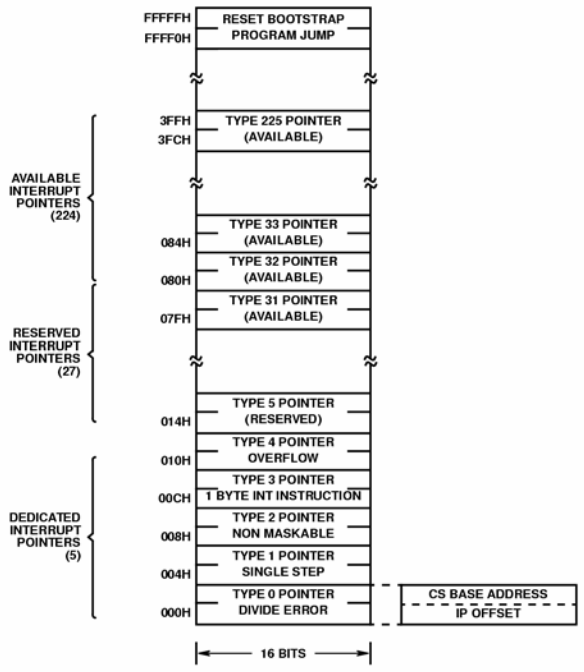
- Gestisce le Interruzioni in modo Vettorizzato
- Ha un pin di NMI
- Ha alcuni segnali di controllo per la gestione di un PIC, Programmable Interrupt Controller
- L'8086 può ricevere 256 diverse interruzioni e pertanto gestisce nei primi 1024 byte della memoria la Vector Table.
- Un tabella in cui sono presenti gli indirizzi della memoria di CODICE delle routine che devono essere eseguite a fronte delle interruzioni.
- Tali indirizzi sono nelle forma, Offset e segmento, pertanto ogni indirizzo e' composto da due word a 16 bit

Paolo Nesi, Univ. Firenze, Italy, 2003-07 397

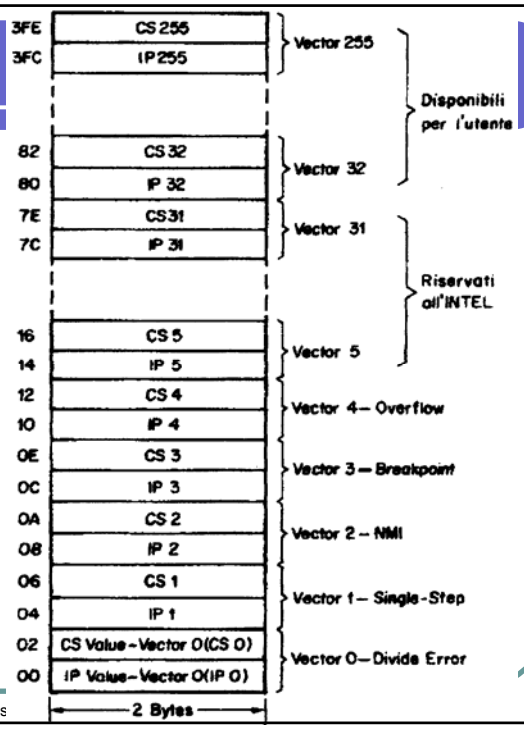
Piedinatura dell'8086



Vettore delle Interruzioni dell'8086



Vettore delle Interruzioni 8086

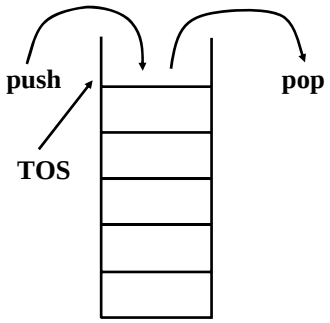


La gestione dello Stack

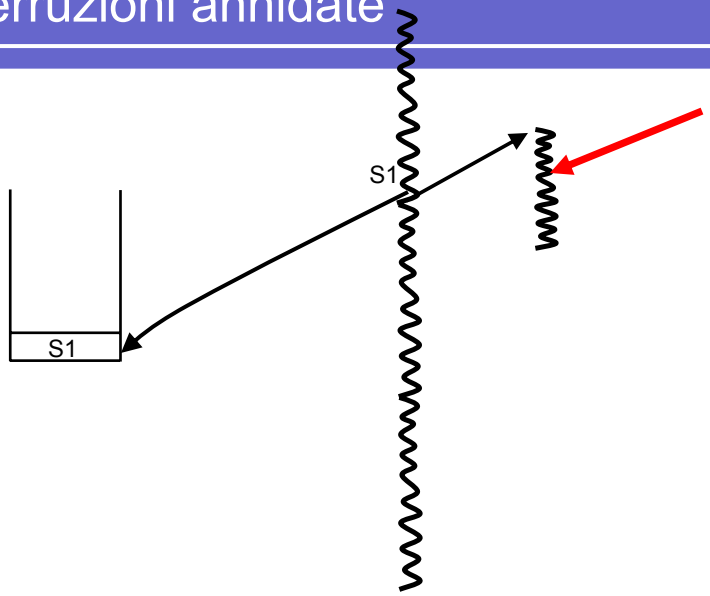
- La pila o *stack* e' un struttura lineare dove le operazioni di inserimento ed estrazione dei dati avvengono sempre dalla stessa parte. La pila e' un struttura che e' governata da una politica di tipo FILO (*first input last output*) il primo a entrare e' l'ultimo a uscire, oppure equivalentemente LIFO (*last input first output*), l'ultimo a entrare e' il primo ad uscire.
- Si prenda ad esempio un tubetto di pasticche di vitamina. Per togliere l'ultima (la prima pasticca che e' stata inserita) e' necessario togliere tutte le pasticche presenti.
- L'operazione di inserimento viene detta di *push* mentre l'operazione di estrazione viene detta di *pop*. Per la sua gestione e' necessario conoscere solamente il punto in cui viene fatta l'inserzione oppure l'estrazione. Tale punto e' detto *Top of the Stack*, TOS.
- Lo *stack* e' tipicamente una struttura dinamica perché le sue dimensioni possono non essere note al momento della realizzazione.

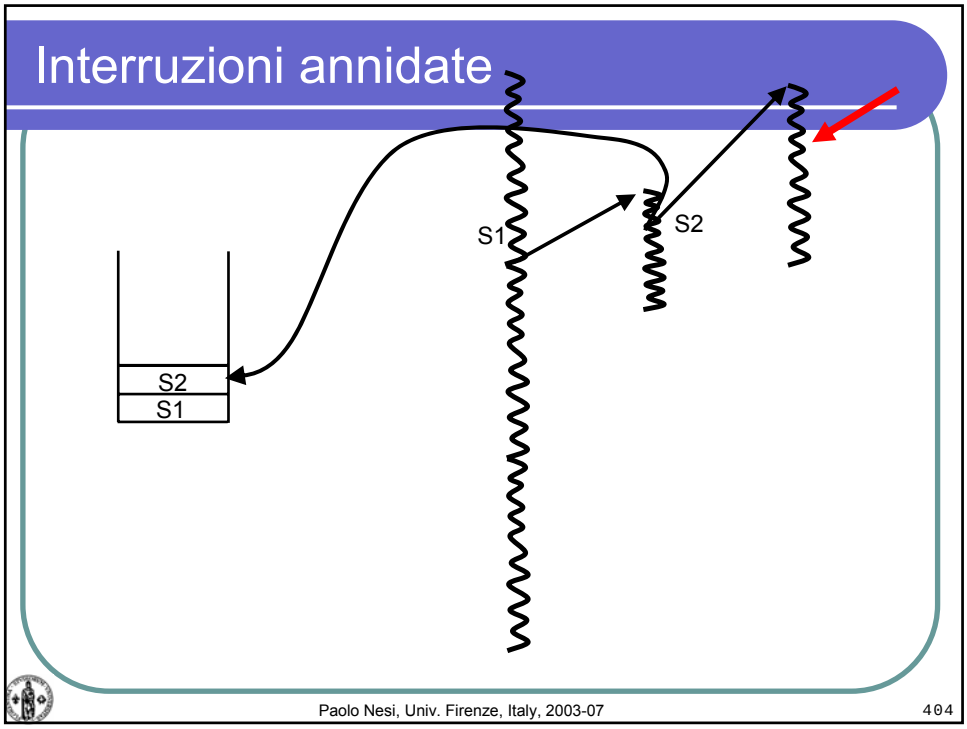
Lo Stack

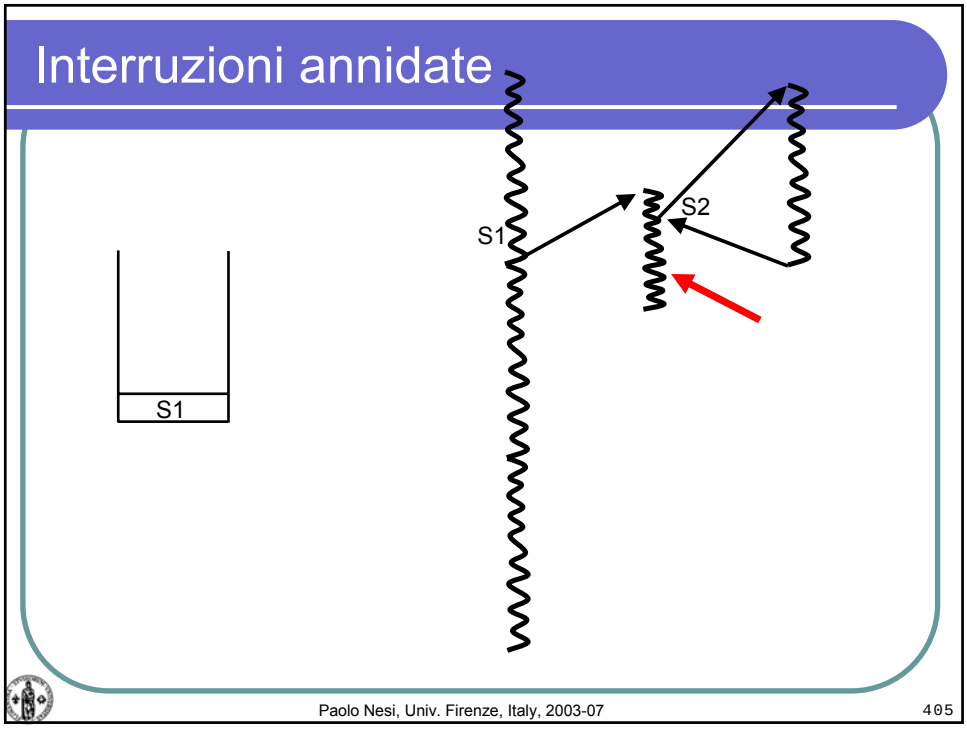
- Poiché tutte le operazioni fanno riferimento ad uno stato iniziale dello *stack*, stato definito alla sua realizzazione, e' necessario tenere sotto controllo quando lo *stack* si svuota.
- Durante la gestione dello *stack* può altrimenti accadere che si cerchi di effettuare un'operazione di pop senza che vi sia un elemento da estrarre dallo *stack*.
- Pertanto le operazioni di base sono la creazione, l'inserimento (pop), l'estrazione (*push*) e la verifica di *stack* vuoto prima di fare ogni *pop*.

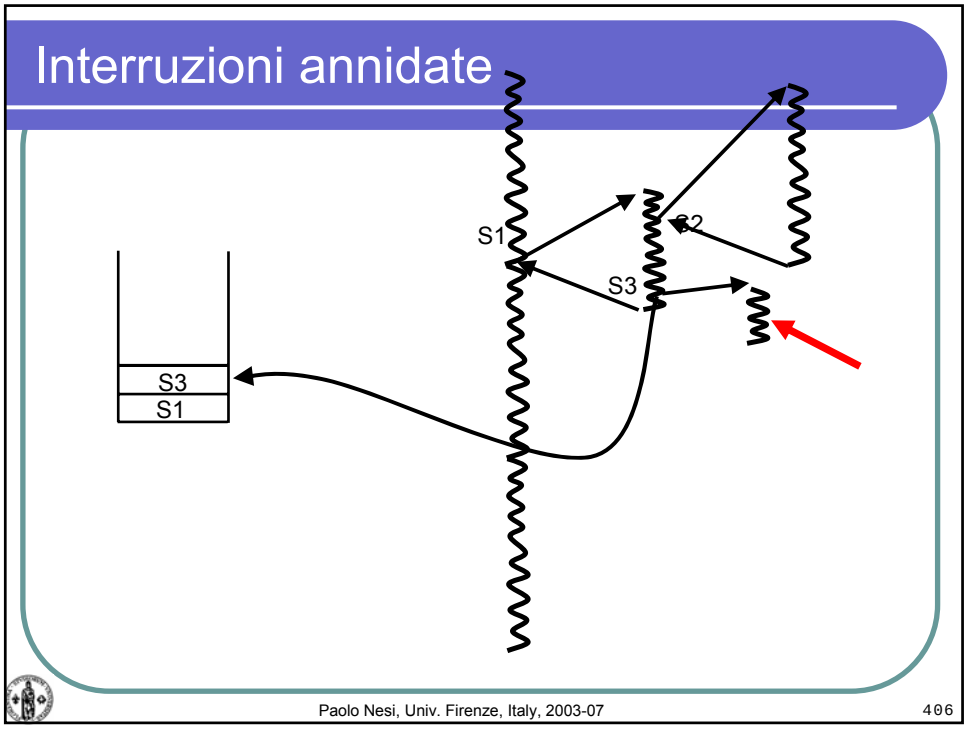


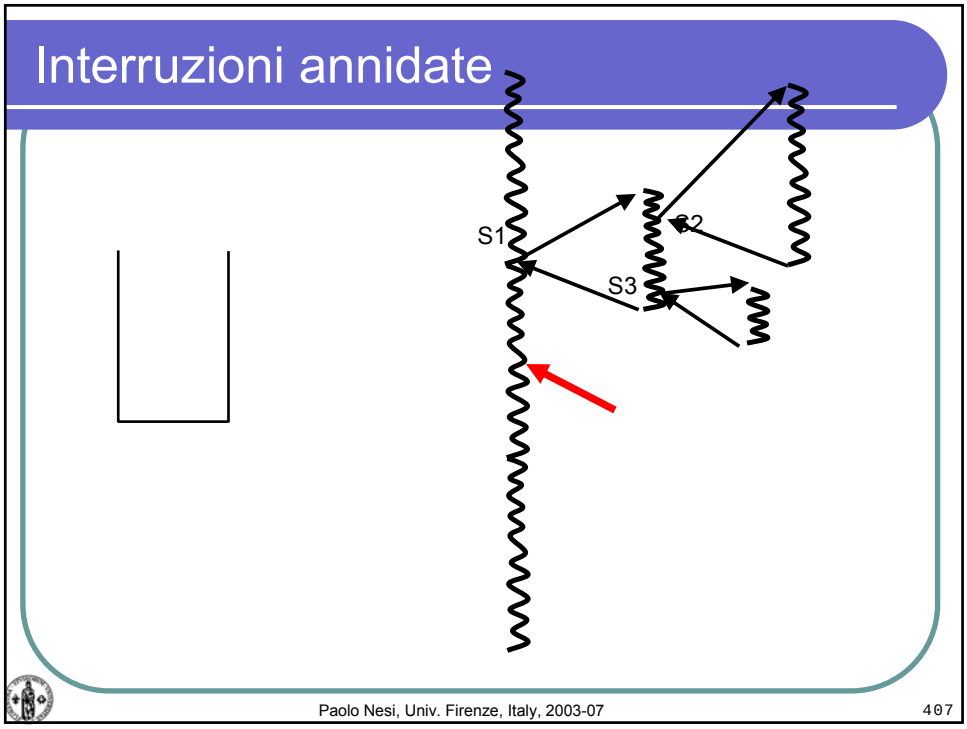
Interruzioni annidate



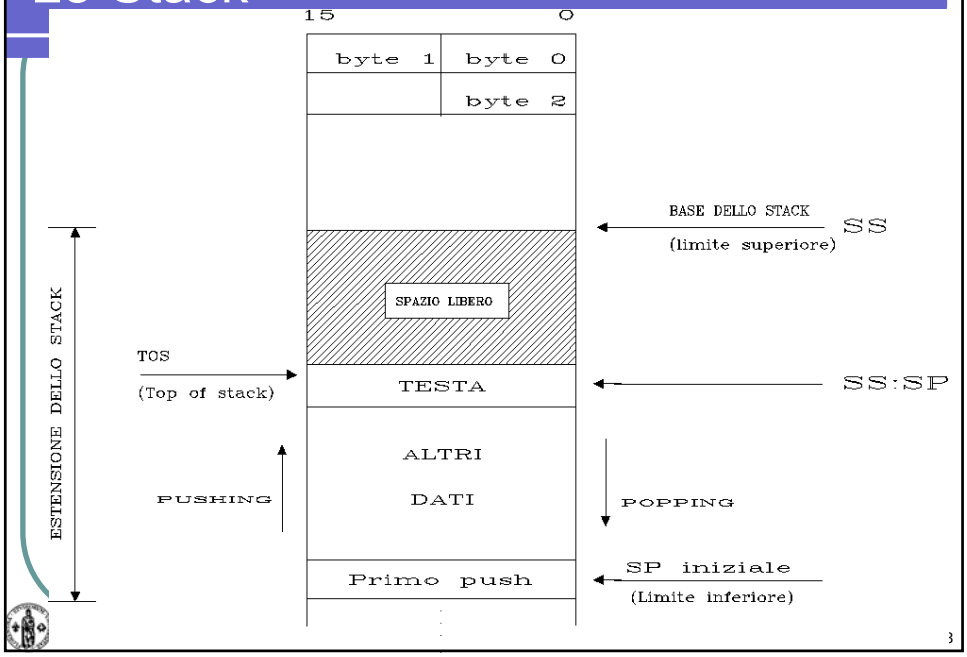

Interruzioni annidate
S1
S2
S2
S1
Paolo Nesi, Univ. Firenze, Italy, 2003-07
404


Interruzioni annidate
S1
S2
S1
Paolo Nesi, Univ. Firenze, Italy, 2003-07
405

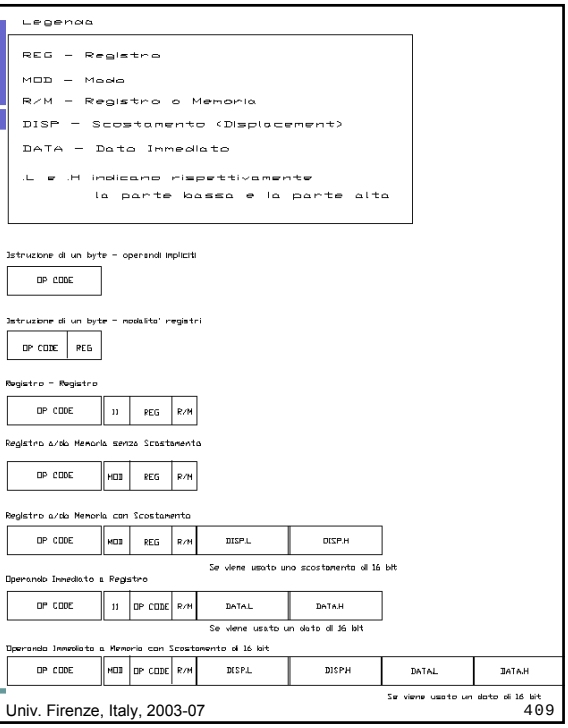

Interruzioni annidate
S1
S2
S3
S3
S1
Paolo Nesi, Univ. Firenze, Italy, 2003-07
406


Interruzioni annidate
S1
S2
S3
Paolo Nesi, Univ. Firenze, Italy, 2003-07
407

Lo Stack



La struttura delle istruzioni dell'8086



Le istruzioni hanno

- Un Opcode che ha praticamente dimensione costante, permette di identificare il tipo dell'istruzione – e.g., MOV, ADD, SUB, etc.
- Dimensioni delle istruzioni con Numero di byte diverso (si veda il manuale dell'8086 per un esempio instruction set), pertanto si ha un fetch dell'istruzione su piu' cicli macchina
- Diverso per
  - diversi modi di indirizzamento
  - calcolo dell'EA
  - Identificazione di registri e/o indirizzi, etc.

Istruzioni e l'Indirizzamento

- L'interpretazione del campo IND può essere differente da macchina (CPU) a macchina (CPU)
- Indirizzo effettivo (EA): Il valore che risulta dal calcolo dell'indirizzo attraverso i componenti espliciti contenuti nell'istruzione

LD RA, VAR
MOV AX, VAR

ST VET(R28), R12
MOV VET(IS), BX

Indirizzamenti

- Vedremo ora alcune modalità di indirizzamento senza fare riferimento esplicito all'8086, ma mostrando vari tipi di istruzioni che si possono trovare su varie CPU diverse
- Le modalità proprie dell'8086 saranno presentate durante le lezioni dedicate alla programmazione dell'8086

Modalità di indirizzamento (dati)

Indirizzamento diretto

LD R1, var ; EA= IND R1:= M[EA]

Indirizzamento relativo ai registri

ST var(R3),R6 ; EA= IND + R3 M[EA]:= R6

Indirizzamento indiretto rispetto ai registri

LD R1, (R2) ; EA= R2 , in alcune CPU si utilizza () in altre []

Mov AH, [BH] ; EA = M[BH], BH contiene l'indirizzo della cella

Indirizzamento relativo ai registri indiciato e scalato

LD R1, var (R2) (Rx) ; EA= IND + R2 + RX*d

d è la dimensione dell'elemento

Modalità di indirizzamento (dati)

Indirizzamento indiretto rispetto ai registri con auto-incremento.

```
LD  R1, (R2)+          ; EA= R2;   R2:= R2 + d
```

Indirizzamento immediato

```
LD  R1, [2346]         ; R1:= M[2346]
LD  R1, 2346           ; R1:= 2346   // questa e' una copia
```

Indirizzamento tra registri, copia fra registri

```
LD  R16,R8             ; R16:= R8
```

Indirizzamento porte di I/O

```
IN   R5,Porta          ; R5:= porta (di ingresso)
```

Istruzioni di controllo

- Possono essere di salto, salto condizionato, chiamata e ritorno da sottoprogrammi
  - Diretto
  - Relativo al PC (CS:IP) o ad altro registro
- Esempi

```
JMP    DEST        ; Diretto o relativo a PC/IP
JZ     wait        ; Di solito relativo a PC/IP
call   sub         ; Di solito diretto
BR     R30         ; EA destinazione = R30
INT 21             ; interruzione
```

In alcuni casi la CPU salva lo stato automaticamente nello Stack in altri no.
Alcune di queste sono per cambi di contesto profondi altre per piccoli cambiamenti dentro lo stesso programma, e.g., dentro il CS nel caso dell'8086.
In ogni caso se si cambia il contesto è possibile utilizzare lo Stack

Breve sommario delle istruzioni

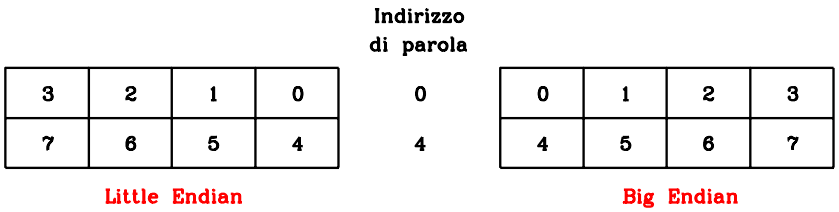
- **Le categorie di istruzioni sono:**
  - Aritmetiche e logiche:
    - ADD, SUB, SHR, SHL, AND, OR
  - Movimento:
    - MOV
    - non 8086: ST, LD
  - Input/Output
    - IN, OUT
  - Cambio di contesto:
    - CALL, JMP, INT, etc.
- Questa carrellata veloce serve solo a darvi un'idea delle istruzioni per procedere poi a vedere i sottositemi di I/O mentre una visione di dettaglio sarà fornita durante le lezioni dedicate alla programmazione dell'8086 in linguaggio Assembly in seguito

Istruzioni 8086

Trasferimento dati	
MOV	Sposta un byte o una parola
PUSH	Impila una parola sullo stack
OUT	Trasferisce un byte o una parola in uscita
LEA	*Load Effective Address*
POPF	Estrae la parola di stato dallo stack
Istruzioni aritmetiche	
ADD	Somma byte o parole
CMP	Compara byte o parole
AAS	Aggiusta ASCII per somma
IMUL	Moltiplica (interi) byte o parole
Manipolazione dei bit	
AND	Esegue l'AND tra due byte o due parole
SHL	Scorrimento a sinistra
RCR	Ruota a destra attraverso il riporto
Manipolazione stringhe	
MOVSB	Sposta una stringa di byte
CMPS	Confronta stringhe
Salti	
JMP	Salto incondizionato
CALL	Chiamata di sottoprogramma
JE	Salta se uguale
JNGE	Salta se non maggiore o uguale
INT	Interruzione (software)
IRET	Ritorno da (routine) interruzione
Controllo della CPU	
STC	Porta a 1 il bit di riporto
CLI	Disabilita il sistema di interruzione
STD	Stabilisce la direzione di scansione stringhe
Sincronizzazione con l'esterno	
HLT	Passa allo stato di *halt*
WAIT	Passa allo stato di *wait*
Istruzione di non operazione	
NOP	Non fa niente

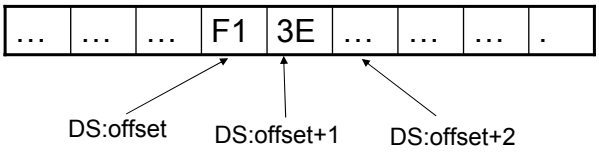
Ordinamento byte in memoria

- Intel: Little Endian
- Motorola: Big Endian
- PowerPC: a scelta



Allineamento dati: esempio 8086

- Se si ha un Intero in rappresentazione complemento a 2: 3EF1H
  - Il byte meno significativo è pari a F1
  - Il suo bit di segno zero, positivo
- In memoria viene allocato come segue:



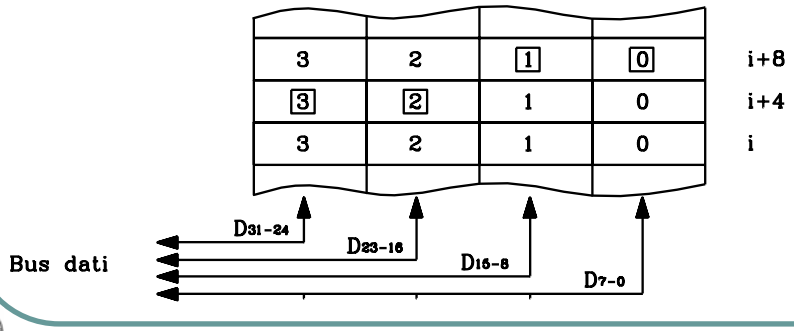
- Lo vedo pertanto in ordine invertito

Effetti del non allineamento, e.g., i386

- Ipotesi di CPU con 32 bit di indirizzo, 32 bit di data bus
- Dati a 32 bit (un ciclo macchina per leggere 4 byte)
- Pertanto con l'allineamento a 32 bit, 4 byte
- Indirizzi Allineati a 4 byte, pertanto 30 bit effettivi, gli altri per scegliere il byte
- Se i dati non sono allineati a indirizzi multipli di 4 byte, per effettuare la lettura della word a 32 si devono fare due cicli macchina

Allineamento in memoria, i386

- Esempio: parole di 32 bit, formate da quattro banchi di 8 byte
- La parola tratteggiata è non allineata; ha il byte meno significativo in i+6 (Little Endian) il più significativo in i+9
- Occorrerebbero due accessi alla memoria



Calcolatori Elettronici

CDL in Ingegneria Elettronica

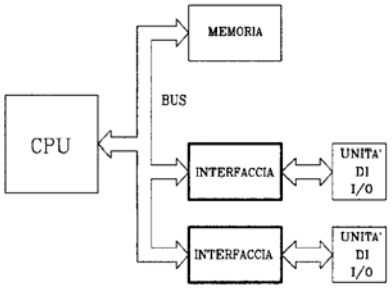
Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

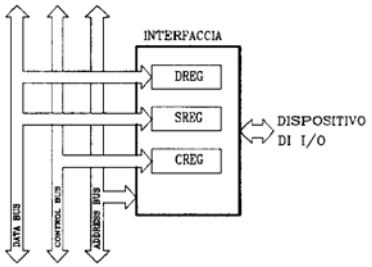
Parte 7, Il Sistema di I/O

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
Agosto, 2003

Schema Generale di I/O



a) Schema di riferimento



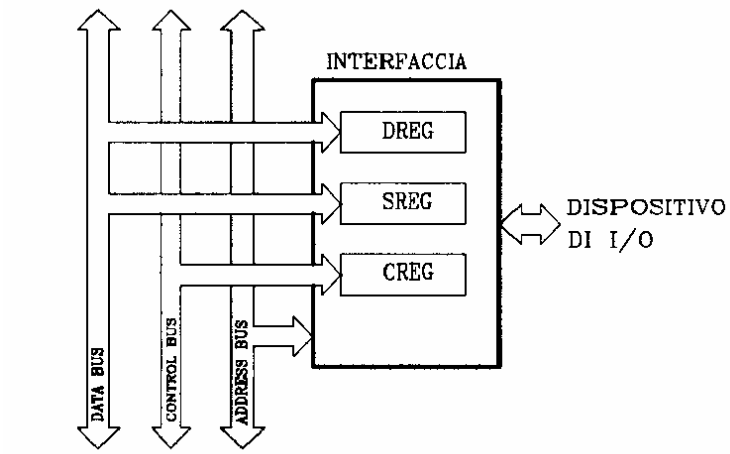
b) Elementi fondamentali di una interfaccia

DREG (Data REGister) - **CREG** (Command REGister) - **SREG** (Status REGister)

Ci sono due modalità per associare un indirizzo a ciascun registro

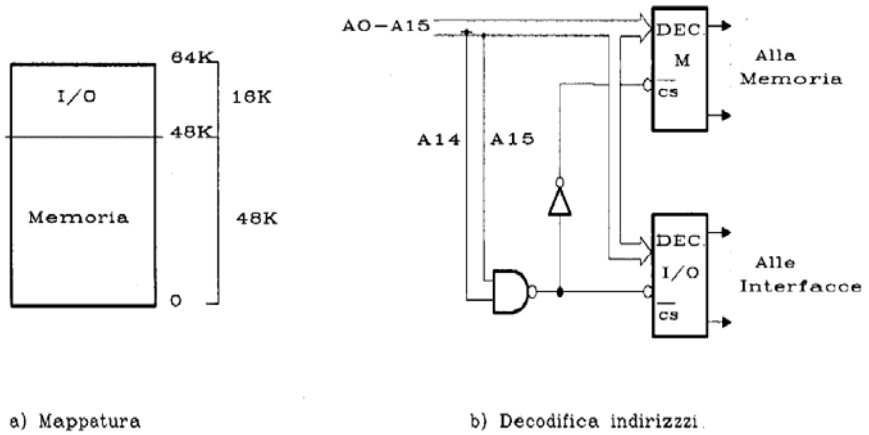
- Ingresso/uscita mappato in memoria
- Ingresso/uscita isolato

Elementi di un'interfaccia



I/O mappato in memoria

- Non sono necessarie istruzioni dedicate per l'I/O



a) Mappatura

b) Decodifica indirizzi

Esempio di istruzioni, circuito precedente

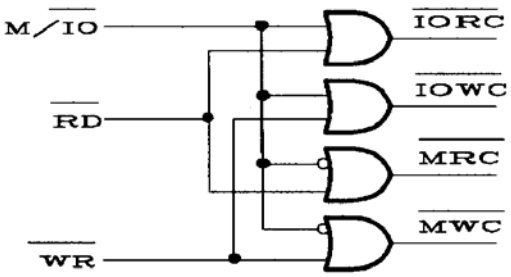
- I/O mappato in memoria

- Mov [FEh],AH ; scrivo in IO
- Mov [03h], AL ; scrivo in Memoria

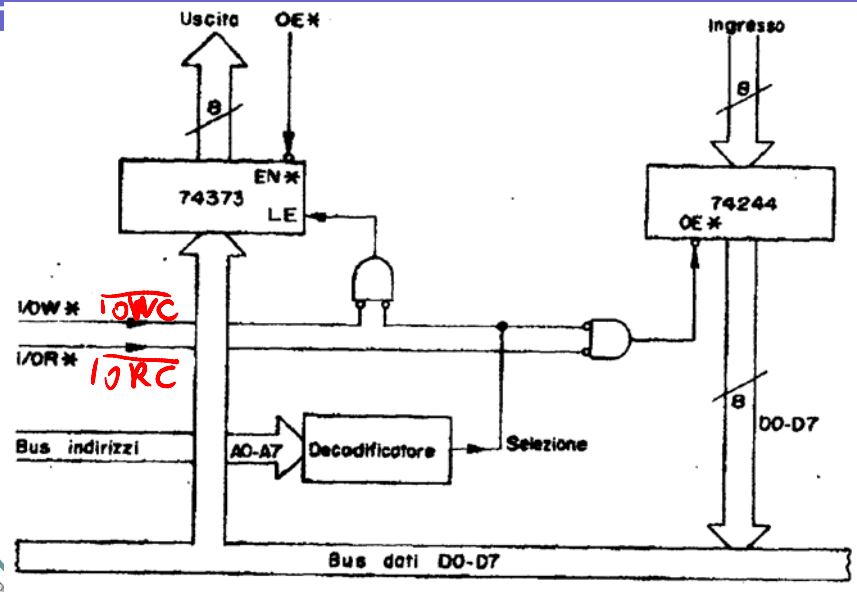
I/O Isolato, Indipendente

- Ci sono istruzioni dedicate per l'I/O
- Due specifiche linee di comando
  - IOWC (I/O Write Command)
  - IORC (I/O Read Command)

In MODO Massimo tali segnali vengo prodotti dal controllore di BUS



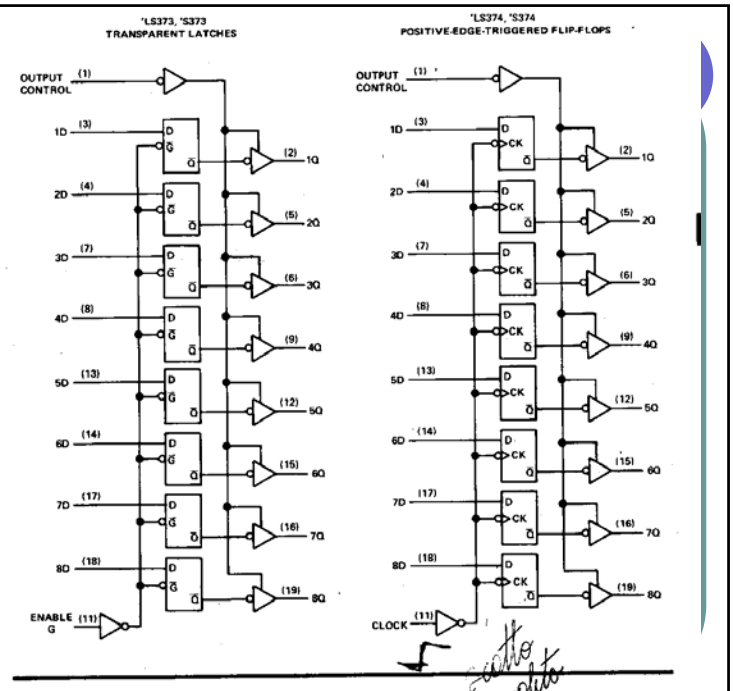
Interfaccia I/O semplice e BUS



Note sull'esempio precedente

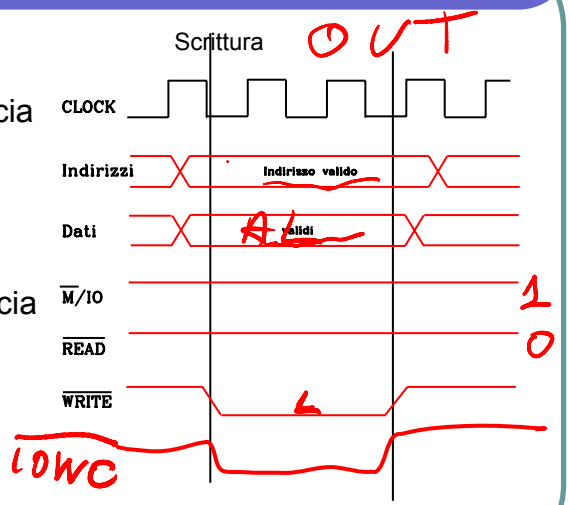
- Il 244 e' stato mostrato in precedenza ed e' un semplice Tri-State Octal Buffer
- Sia la lettura che la scrittura sono allo stesso indirizzo identificato dalla decodifica che vedremo in seguito
- Vi sono istruzioni diverse per la lettura e la scrittura su I/O, queste producono segnali di Ingresso e Uscita rispettivamente

373
e
374



Esempio

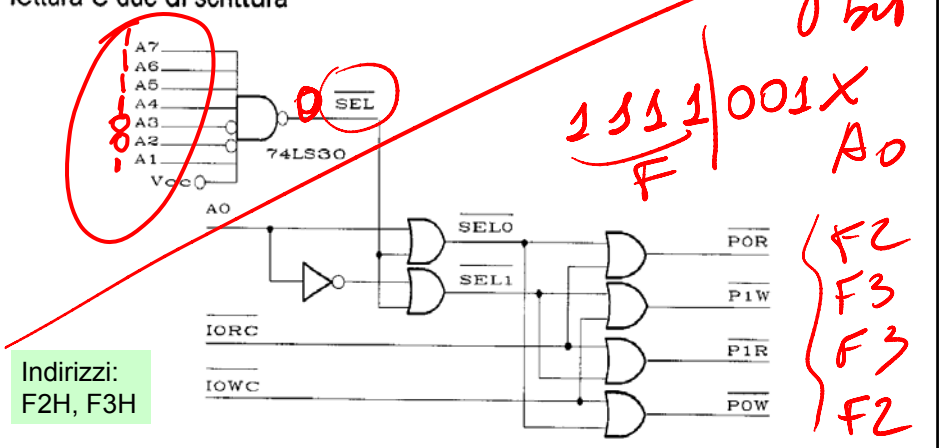
- IN ah,32h
  - leggo dall'interfaccia
- OUT 32h, AL
  - scrivo sull'interfaccia



Paolo Nesi, Univ. Firenze, Italy, 2003-07
432

Decodifica degli indirizzi

- Sull'interfaccia deve essere presente una parte di decodifica degli indirizzi
- Esempio di decodifica di comandi e indirizzi per un'interfaccia con due porte di lettura e due di scrittura



Indirizzi:
F2H, F3H

Modalità di Esecuzione di op. I/O

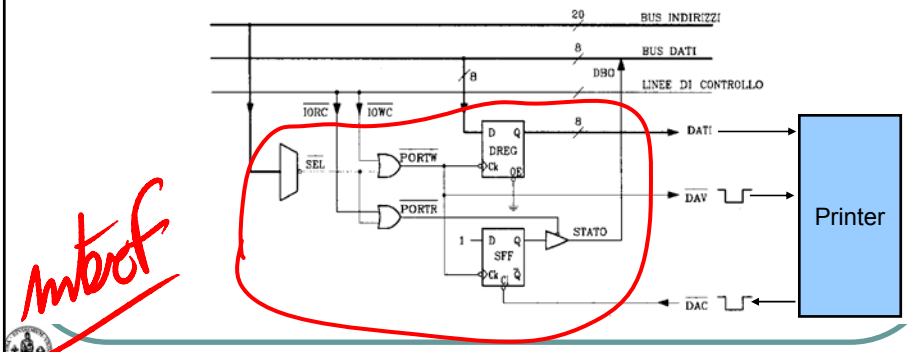
- Per la sincronizzazione tra programma in esecuzione e dispositivi di I/O è necessario tener conto della diversa velocità di CPU e dispositivi esterni
- Ci sono tre principali approcci:
  - Controllo di programma
  - Controllo di interruzione
  - Accesso diretto alla memoria (DMA) o con processori di I/O

- Protocollo di Comunicazione fra CPU e Periferica

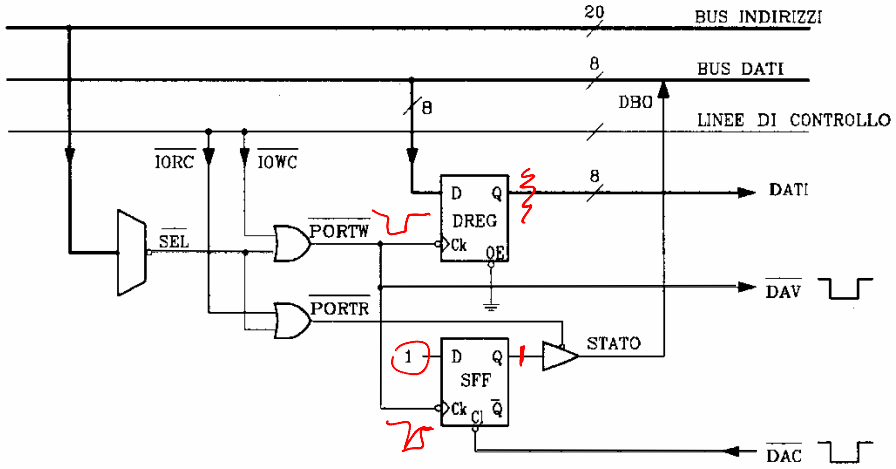
Gestione a Controllo di Programma, interfaccia

Esempio: trasferimento di un blocco di dati verso una stampante

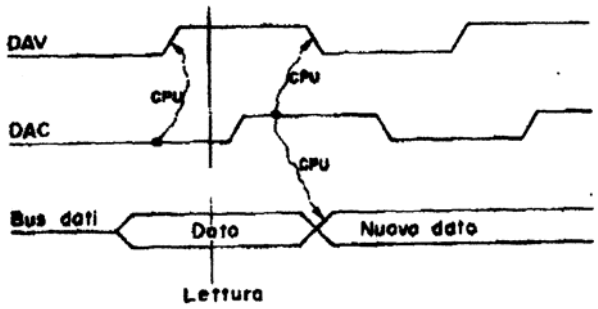
- $\overline{DAV}$ indica la disponibilità di un nuovo carattere all'ingresso della stampante
- $\overline{DAC}$ indica che la stampante ha terminato la stampa del carattere
- L'interfaccia contiene un indicatore dello stato della stampante



Gestione a Controllo di Programma, interfaccia



Evoluzione temporale dei segnali DAV e DAC



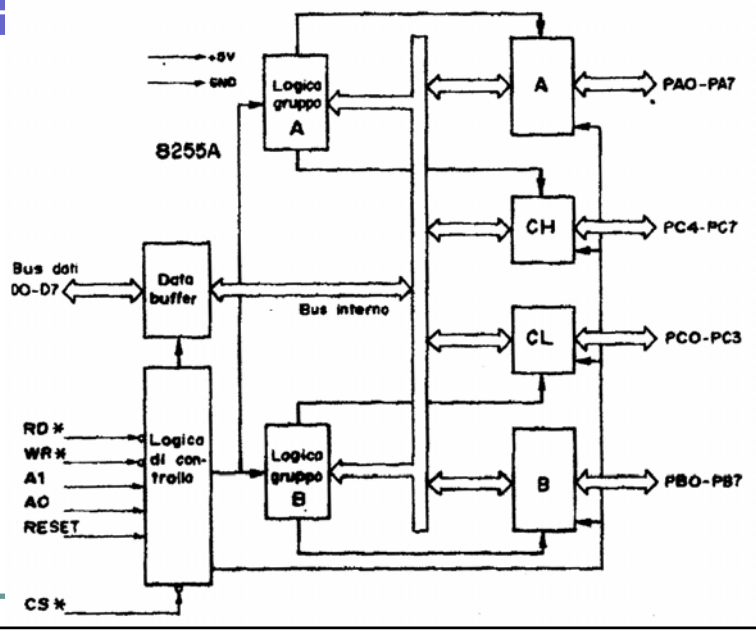
- Meccanismo di Handshaking fra CPU ed una periferica in lettura
- In questa figura DAC e DAV sono attivi alti
- Con DAV alto la CPU comunica che il dato e' presente, questo viene effettuato solo se il DAC e' basso
- Dopo la lettura la periferica mette DAC alto, questo viene letto dalla CPU che mette giù il DAV e e un nuovo dato valido
- ….

Gestione a Controllo di Programma

L'istruzione OUT PORT, AL copia il contenuto di AL in DREG dell'interfaccia, porta SFF a 1 e trasmette il $\overline{DAV}$ alla stampante

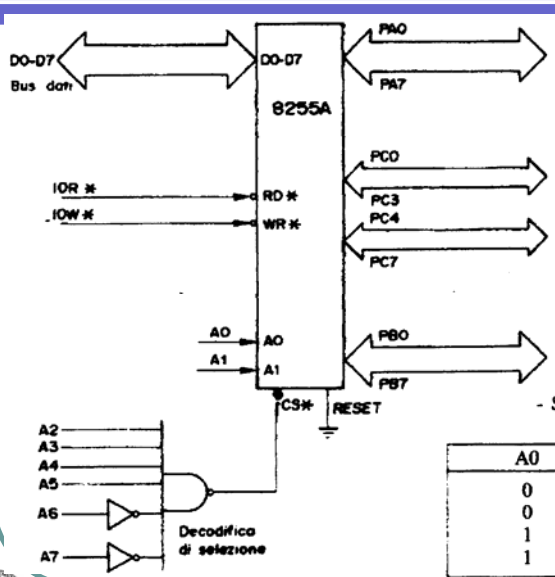
```
STAMPA:  MOV   AL, SI
         OUT   PORT, AL
ATTESA:  IN    AL, PORT
         AND   AL, 1
         JNZ   ATTESA
         INC   SI
         LOOP  STAMPA
         RET
```

I/O con l'8255A



439

Collegamento dell'8255



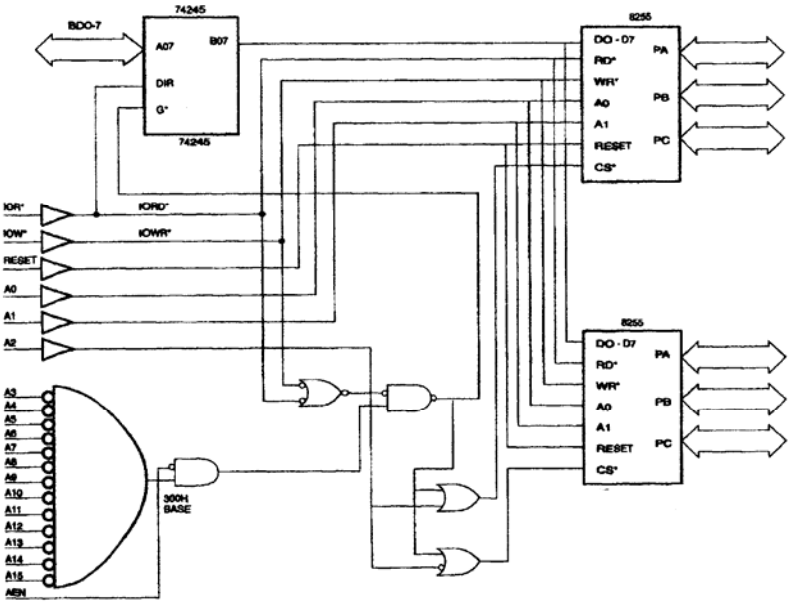
- SELEZIONE DELLE PORTE A, B, C E DEL REGISTRO R DELL'8255A

A0	A1	Porta
0	0	A
0	1	B
1	0	C
1	1	R

Paolo Nesi, Univ. Firenze, Italy, 2003-07

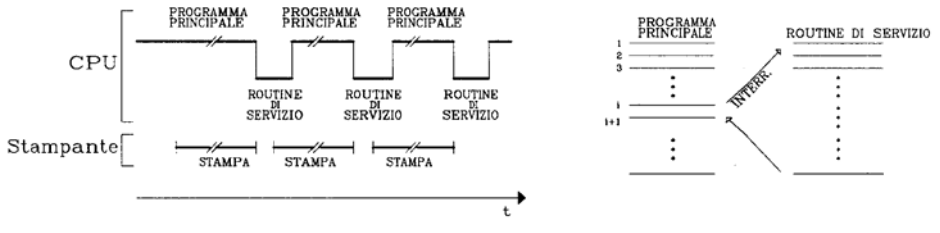
440

Esempio di Interfaccia con 8255

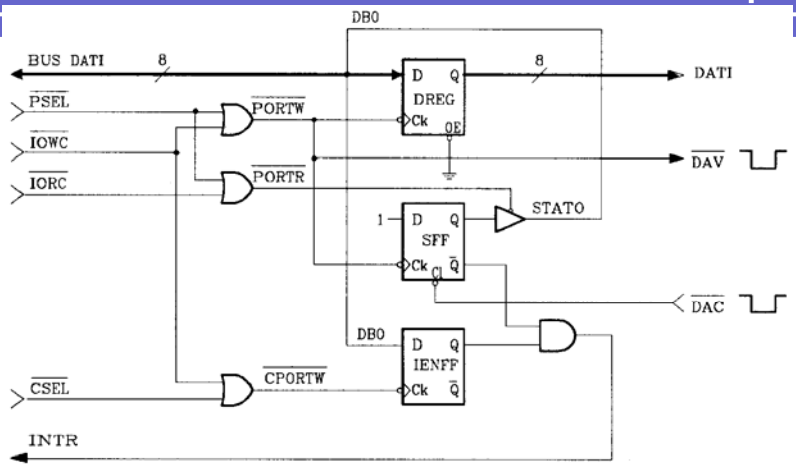


Gestione delle Interruzioni

- Le interruzioni consentono di proseguire l'esecuzione di un programma mentre viene effettuta l'operazione di I/O dalla periferica
- Un segnale INTR generato dalla periferica indica quando questa ha terminato l'operazione richiesta
- A seguito della ricezione di una richiesta di interruzione, viene eseguita una routine di servizio (*interrupt handler*)



Interfaccia di Uscita con Interrupt



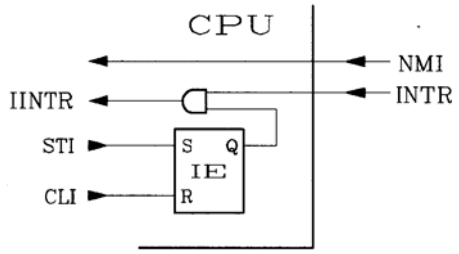
- L'interfaccia deve generare la richiesta di interruzione
- PORTW disasserisce INTR, $\overline{DAC}$ lo asserisce

Mascheramento delle Interruzioni

- È possibile mascherare (disabilitare) la richiesta di interruzione tramite il flip-flop IENFF

```
MOV     AL, 1/0      ; di Set/Reset
OUT     CPORT, AL    ; CPORT: porta controllo
```

- È posssibile mascherare l'interruzione internamente alla CPU



- Ci sono anche interruzioni non mascherabili NMI dell'8086

Routine/Procedura di Servizio

La routine di servizio dell'interruzione deve compiere alcune operazioni fondamentali

- Salvare i registri della CPU (in particolare PSW: Program Status Word)
- Servire la periferica
- Disasserire la richiesta di interruzione
- Rirpristinare le informazioni salvate al primo punto
- Ritornare al programma originale nel punto in cui era stato interrotto
  - Ripristinando il valore del PC originale
  - Riabilitando il flip-flop IE della CPU

Interruzioni da parte di più periferiche

Se si hanno più periferiche in grado di generare le interruzioni si hanno i seguenti problemi

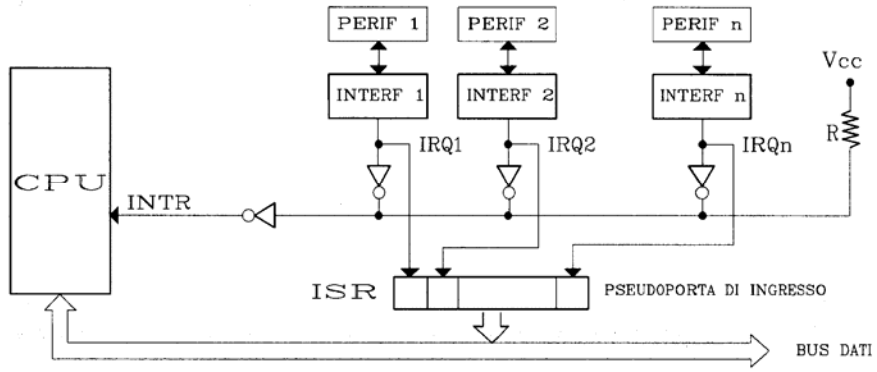
- Individuazione della periferica
- Scelta della routine di servizio
- Priorità fra periferiche (interruzioni)
- Interrompibilità della routine di servizio

E possibile:

- Gestire le richieste da programma
- Impiegare interruzioni vettorizzate
- Impiegare uno schema *daisy chain*

Gestione a Polling, Interfaccia

Le istruzioni possono essere discriminate leggendo una porta (ISR) i cui bit rappresentano lo stato delle singole IRQ

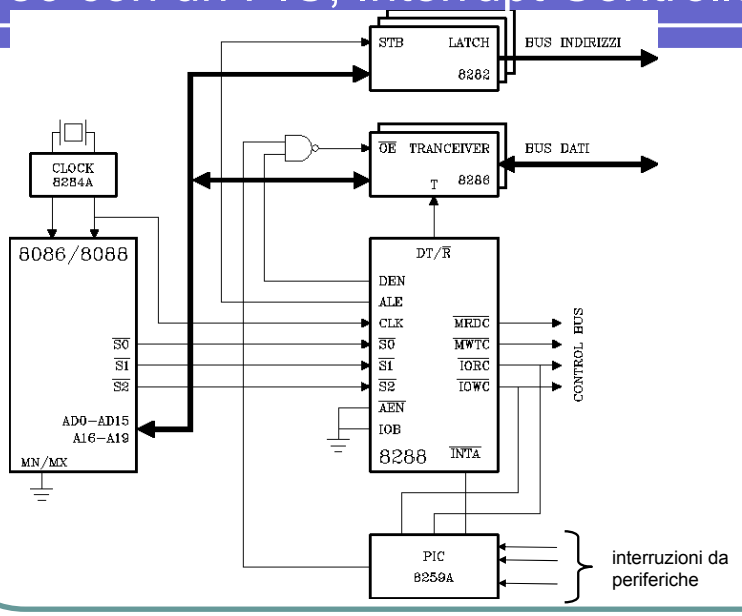


Si parla di priorità software

Polling, programma

```
IN   AL,ISPORT ; lettura di ISR
MOV  COPIA,AL
AND  1 ; maschera per verificare bit 0
JNZ  ROUT0 ; salto alla routine per int 0
MOV  COPIA,AL
AND  2 ; maschera per verificare bit 1
JNZ  ROUT1 ; salto alla routine per int 1
MOV  COPIA,AL
AND  4 ; maschera per verificare bit 2
JNZ  ROUT2 ; salto alla routine per int 2
 .
 .
MOV  COPIA,AL
AND  80H ; maschera per verificare bit 7
JNZ  ROUT7 ; salto alla routine per int 7
```

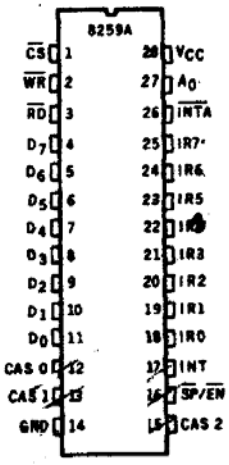
8086 con un PIC, Interrupt Controller



Paolo Nesi, Univ. Firenze, Italy, 2003-07 450

L'8259, Programmable Interrupt Controller

Disposizione dei pin

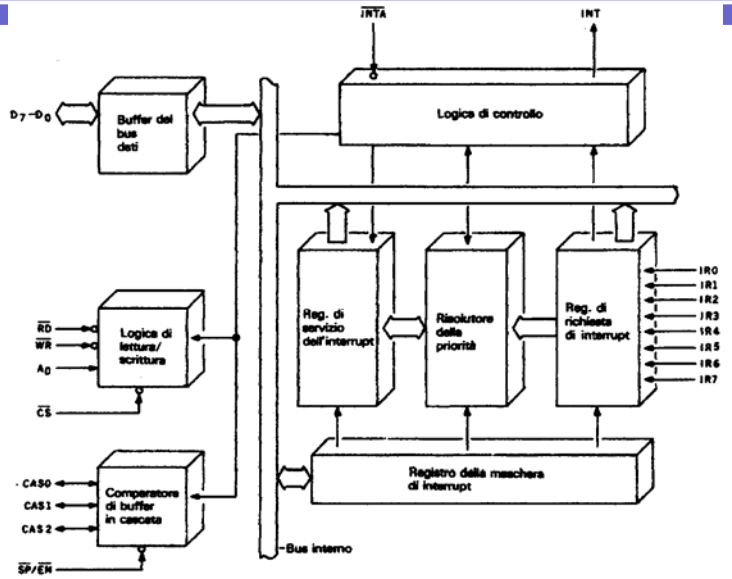


Nomi dei pin

$D_7 - D_0$	Bus dati (bidirezionale)
$\overline{RD}$	Lettura input
$\overline{WR}$	Scrittura input
A_0	Indirizzo di selezione del comando
$\overline{CS}$	Selezione del chip
CAS2 CAS0	Linee in cascata
$\overline{SP/EN}$	Buffer di abilitazione del progr. secondario
INT	Output dell'interrupt
$\overline{INTA}$	Input di riconoscimento dell'interrupt
IR0–IR7	Input delle richieste di interrupt

Paolo Nesi, Univ. Firenze, Italy, 2003-07 451

La struttura del PIC 8259



Struttura interna e disposizione dei pin del controllore programmabile di interrupt 8259

452

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 8, Altri Aspetti

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
2003-2007

BUS: Master vs Slave

- Un'operazione di trasferimento su un bus coinvolge due fasi:
  - inviare la richiesta di trasferimento (il comando)
  - trasferire i dati
- Il **master** è il dispositivo che avvia l'operazione di trasferimento (ad es. il processore)
  - commando di read o write
- Lo **slave** è il dispositivo che risponde alla richiesta (ad es. la memoria)
  - invia o riceve i dati

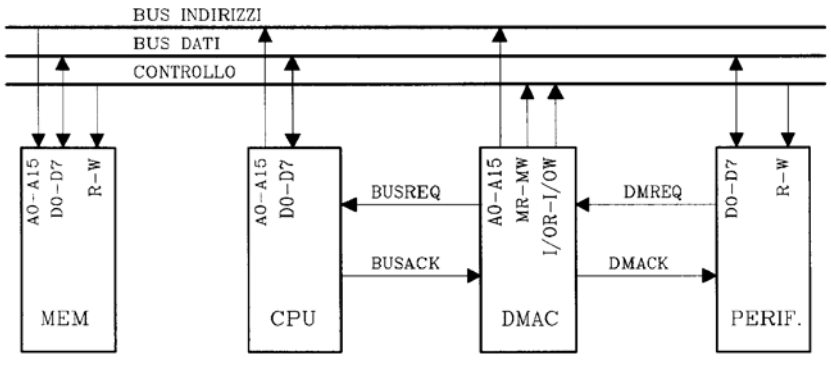
DMA - Direct Memory Access

- Il DMA controller è un processore specializzato nel trasferimento dei dati tra dispositivi di I/O e la memoria principale e viceversa senza l'intervento del processore.
- A fronte di una richiesta di I/O da parte di un programma, il processore tramite il *device driver* invia al DMA controller le seguenti informazioni per programmare il DMA controller:
  - Tipo di operazione richiesta lettura o scrittura
  - Indirizzo di memoria da cui iniziare a leggere/scrivere i dati
  - Numero di byte da leggere/scrivere

DMA - Direct Memory Access

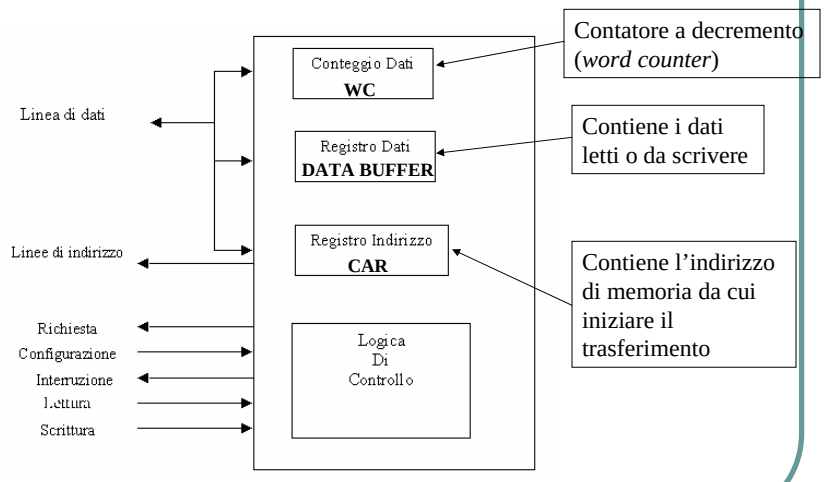
- Il processore si svincola completamente dall'esecuzione dell'operazione di I/O.
- Il DMA controller avvia l'operazione richiesta e trasferisce i dati da/verso la memoria.
- Dopo aver trasferito tutti i dati, il DMA controller invia un interrupt al processore per segnalare il completamento dell'operazione richiesta.
- Modalità di trasferimento possibili:
  - burst transfer;
  - cycle stealing;
  - transparent DMA.

DMA, Direct Memory Access



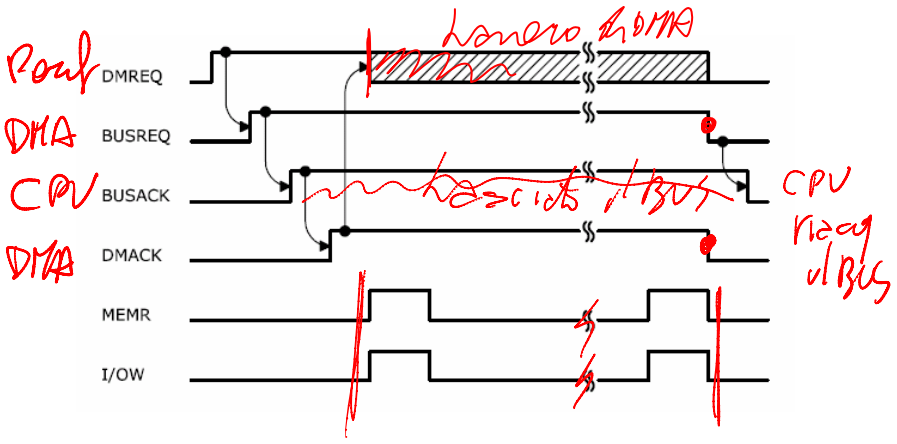
- DMREQ. Richiesta di trasferimento
- BUSREQ. Richiesta di possesso del bus
- DMACK, BUSACK. Richieste accolte

DMA - Logica



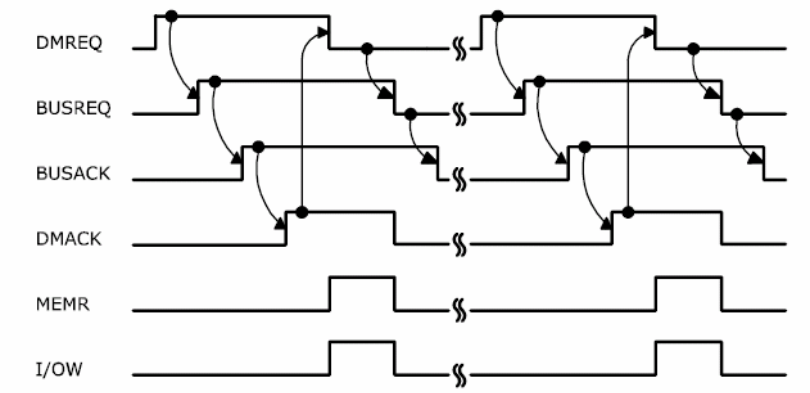
Paolo Nesi, Univ. Firenze, Italy, 2003-07 458

Ciclo di acquisizione del BUS



Paolo Nesi, Univ. Firenze, Italy, 2003-07 459

Trasferimento tramite DMA



DMA - Direct Memory Access

Burst mode:

- Il DMA acquisisce il controllo del bus;
- Il DMA rilascia il bus solo quando ha terminato il trasferimento.

- Vantaggio: il trasferimento avviene alla massima velocità possibile.
- Svantaggio: la CPU non può accedere al bus durante il trasferimento.

DMA - Direct Memory Access

Cycle stealing:

- Il dati sono trasferiti in piccoli blocchi;
- Il DMA controller diventa padrone del bus per brevi istanti di tempo.

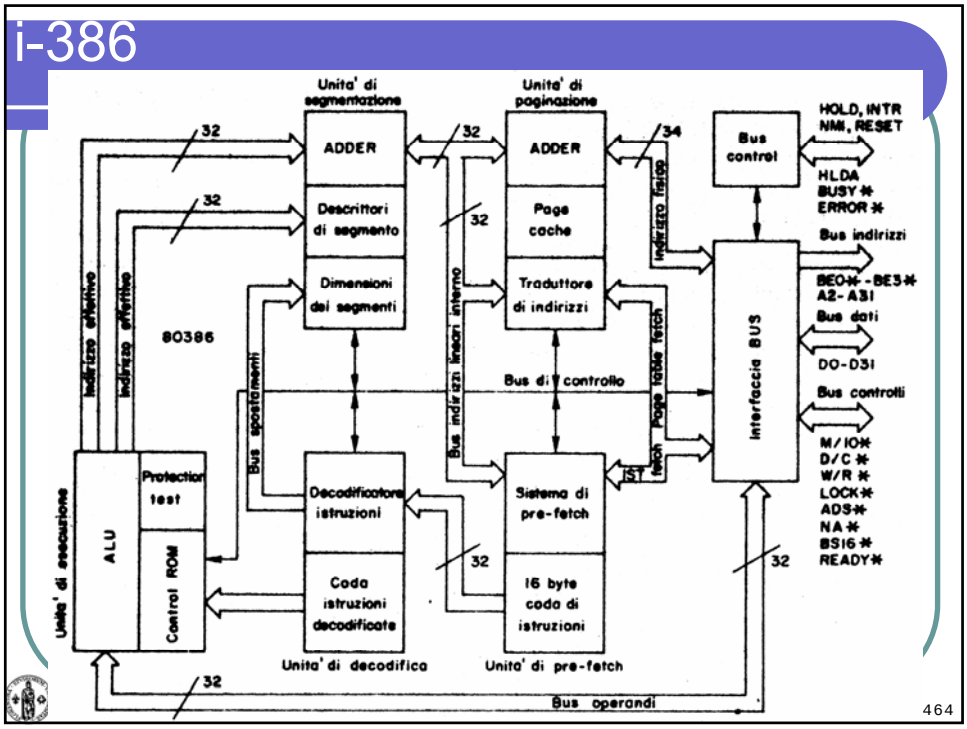
- Vantaggio: la CPU non è bloccata per lunghi istanti di tempo.
- Svantaggi: il trasferimento richiede più tempo.

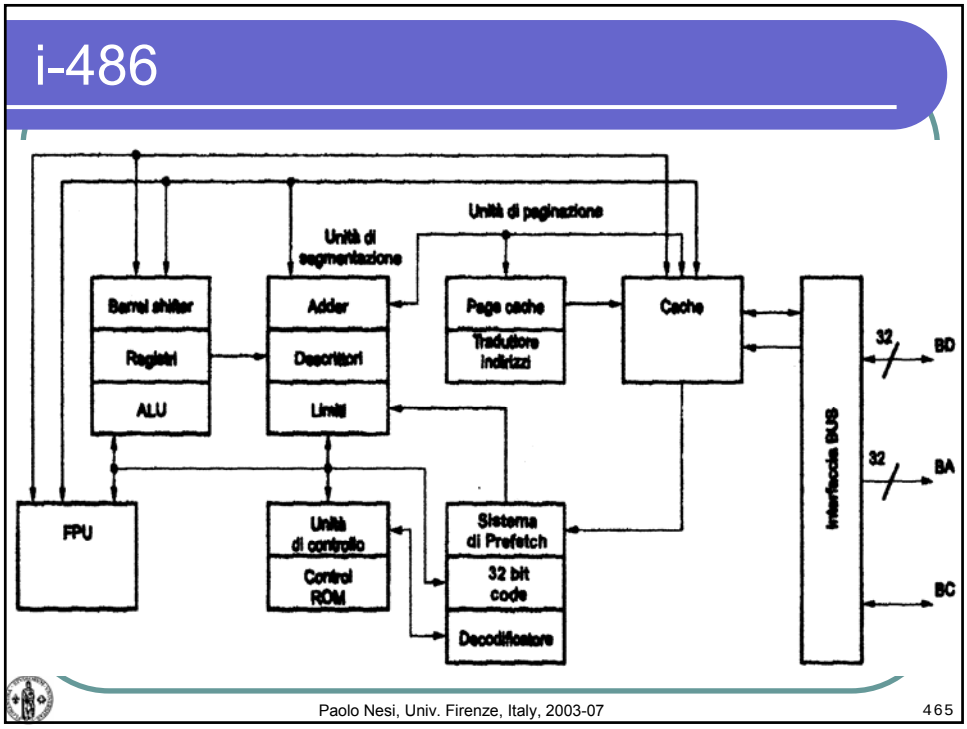
DMA - Direct Memory Access

Transparent DMA:

- Il DMA rileva quando la CPU non usa il bus;
- I trasferimenti hanno luogo solo quando la CPU non sta usando il bus.

- Vantaggi: il DMA non rallenta mai la CPU.
- Svantaggi: è in media il metodo più lento.


i-386
Unita' di segmentazione
Unita' di paginazione
ADDER
ADDER
Bus control
HOLD, INTR
NMI, RESET
HLDA
BUSY*
ERROR*
32
32
32
34
Descrittori di segmento
Page cache
Dimensioni dei segmenti
Traduttore di indirizzi
Bus indirizzi
BE0*-BE3*
A2-A31
Bus dati
D0-D31
Bus controlli
M/IO*
D/C*
W/R*
LOCK*
ADS*
NA*
BS16*
READY*
80386
Indirizzo effettivo
Indirizzo effettivo
Bus spostamenti
Bus indirizzi lineari interno
Page table fetch
indirizzo fisico
Bus di controllo
interfaccia BUS
Protection test
ALU
Control ROM
Unita' di esecuzione
Decodificatore istruzioni
Coda istruzioni decodificate
Sistema di pre-fetch
16 byte coda di istruzioni
IST fetch
Unita' di decodifica
Unita' di pre-fetch
32
32
Bus operandi
464


i-486
Unità di paginazione
Unità di segmentazione
Barrel shifter
Registri
ALU
Adder
Descrittori
Limiti
Page cache
Traduttore indirizzi
Cache
FPU
Unità di controllo
Control ROM
Sistema di Prefetch
32 bit code
Decodificatore
Interfaccia BUS
32
BD
32
BA
BC
Paolo Nesi, Univ. Firenze, Italy, 2003-07
465

Calcolatori Elettronici

CDL in Ingegneria Elettronica
Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 8bis, La Programmazione

Prof. Paolo Nesi
Ing. Paolo Vaccari
http://www.dsi.unifi.it/~nesi

2007

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*)

La Programmazione

- È l'attività con cui si predispone l'elaboratore a **eseguire un *particolare insieme di azioni* su *particolari dati***, allo scopo di *risolvere un problema.*

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.*
  - *….*

Risoluzione dei problemi

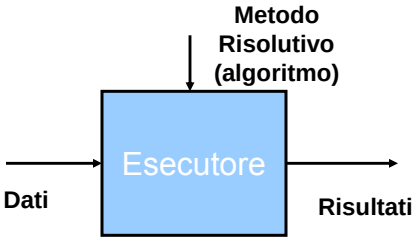
- La descrizione del problema non fornisce (in generale) un metodo per risolverlo.
  - Affinché un problema sia risolvibile è però necessario che la sua definizione sia chiara e completa
- Non tutti i problemi sono risolvibili attraverso l'uso del calcolatore. Esistono classi di problemi per le quali 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

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.*

Algoritmo

- Un algoritmo è una sequenza **finita** di mosse che risolve ***in un tempo finito*** una *classe* di problemi.
- L'esecuzione delle azioni *nell'ordine specificato dall'algoritmo* consente di ottenere, a partire dai dati di ingresso, i risultati che risolvono il problema.



ESECUTORE
una ***macchina astratta***
capace di ***eseguire le azioni***
specificate dall'algoritmo.

Algoritmi: proprietà

- **Eseguibilità**: ogni azione deve 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.

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.

Algoritmi e programmi

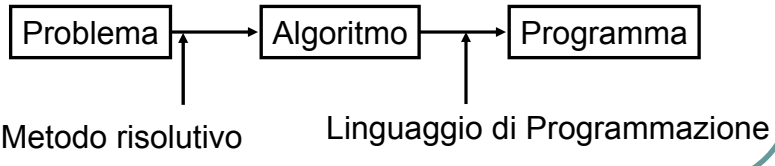
- Ogni elaboratore è una macchina in grado di eseguire azioni elementari su oggetti detti **DATI**.
- L'esecuzione delle azioni è richiesta all'elaboratore tramite comandi elementari chiamati **ISTRUZIONI** espresse attraverso un opportuno formalismo: il **LINGUAGGIO di PROGRAMMAZIONE**.
- La formulazione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

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*.

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**



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!**

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*

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.

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 formalizzare degli algoritmi.

Linguaggi di Programmazione

I linguaggi sono classificati in:

- Low level
  - ⇒ Linguaggio macchina & Linguaggio Assembly
- High level
  - ⇒ C/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)
  - Poi esistono i macro assemblatori….
- I linguaggi *high level* sono più "vicini" al nostro modo di pensare
  - Devono essere opportunamente tradotti per essere compresi da un computer

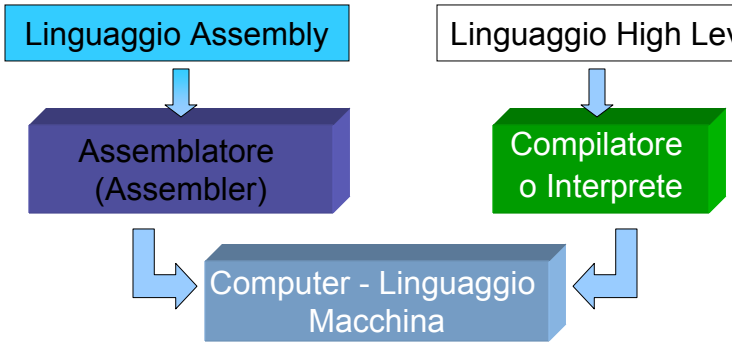
Assembly

- 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.

Relazione tra linguaggi di programmazione

```
START: MOV AX, BX
       CMP AX, 12h
       JZ EQUAL
       INT 21h
       RET
EQUAL: MOV BL, 82h
```

```
void main ( ){
   int x, y;
   printf("Hello world\n");
   if(x>=0)
     stampa(y);
  else
     x++;}      // C/C++
```



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, CPU/OS).
  - 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.

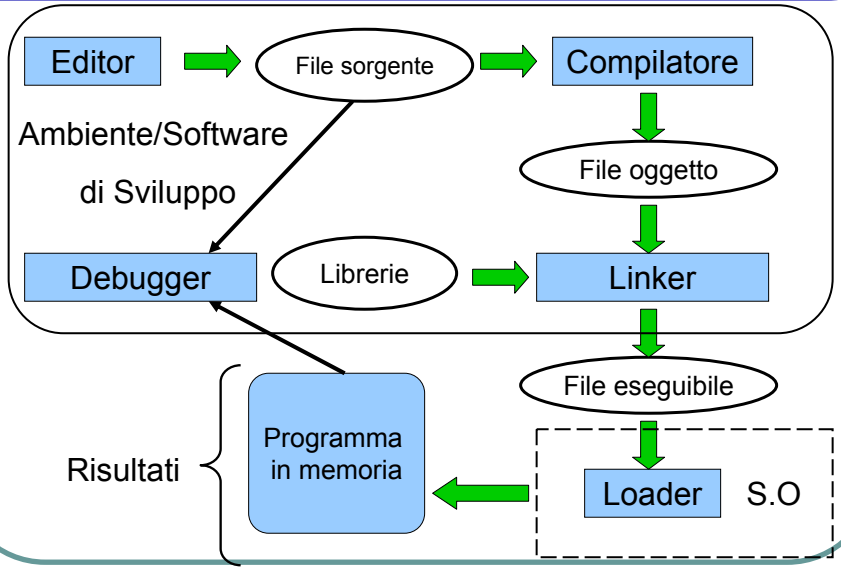
Compilatore vs Interprete

- 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.

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.

Ambiente di sviluppo



Paolo Nesi, Univ. Firenze, Italy, 2003-07 490

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.
- Verifica e debug:
  - Testing, collaudo → identificazione di errori di varia natura
    - Riconduzione degli errori alla causa
    - Controllo passo passo, debug
    - Risoluzione dell'errore via editing, ritorno alla prima fase…

Paolo Nesi, Univ. Firenze, Italy, 2003-07 491

Calcolatori Elettronici

CDL in Ingegneria Elettronica
Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 9, La Programmazione in Assembly

Prof. Paolo Nesi
Ing. Paolo Vaccari
http://www.dsi.unifi.it/~nesi

2007

Quale Assembly?

- Ogni *architettura* (o famiglia di CPU, come ad esempio: Intel x86, PowerPC, SPARC) ha un proprio Assembly
- I concetti di base dell'Assembly sono simili in tutte le architetture…
- Useremo l'Assembly x86
  - Ci sono però due tipi di sintassi differenti:
    - Intel: `mov ebx, 0ffh`
    - AT&T: `movl $0xff, %ebx`
  - Useremo la sintassi Intel

Linguaggio Assembly x86 (1)

- Formato di una linea di codice (*statement*):

  [Nome] [Cod. OP.] [Operandi] [;Commento]
  - **Nome**: identifica *simboli*
  - **Cod. OP.**: identifica *istruzioni* e *direttive* (tramite "mnemonici")
  - **Operandi**: dipendono dal Cod. OP.
  - **Commento**: importantissimo!
- Esempi:

```
mov   ax,bx     ; operazione su registri
var   dw     0  ; alloca una "variabile"
; ricordare: i commenti sono importanti!
ret
```

Linguaggio Assembly x86 (2)

- Due tipi di operazioni:
  1. *Direttive (pseudo-operazioni)*
     - danno informazioni all'assemblatore
     - non sono tradotte in codice macchina
     - esempi:

```
DB  ?                ; alloca un byte
MIO_NUMERO  EQU  12  ; una "costante"
```

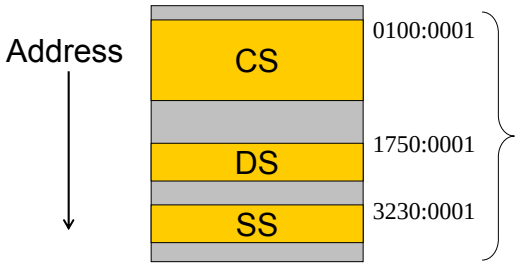
  2. *Istruzioni*
     - sono tradotte in istruzione codice macchina dall'assemblatore
     - esempi:

```
START:  MOV  AX,BX  ; istruz. con "etichetta"
```

Definizione segmenti (1)

- Un programma assembly può essere diviso in vari *segmenti*:
  1. Code segment (CS), definisce il programma principale => l'inizio delle istruzioni.
  2. Data segment (DS), definisce l'inizio dell'area dei dati che verrano usati
  3. Stack segment (SS), definisce lo Stack
  4. Extra segment (ES), può essere usato come segmento ausiliario
- I segmenti possono essere allocati in zone diverse di memoria e lontane tra loro

Definizione segmenti (2)



- Tutti i programmi sono costituiti da uno o più segmenti
- Durante l'esecuzione i registri di segmento puntano ai segmenti attivi
- Registri di segmento a 16 bit, quindi 64Kbyte di dimensione massima del segmento
- Ricordare: il valore del segmento va moltiplicato per 16 (10h)!

Registri segmento

- Quando il sistema operativo 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
```

- Nota: l'indirizzamento immediato non può essere usato per cambiare il valore dei segmenti

Direttiva `segment` (1)

```
<Segment_Name> segment (<align>) (<combine>) (<class>)
  <code>
<Segment_Name> ends
```

- 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

Direttiva **segment** (2)

- Tipicamente ci sono tre 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
```

segment: il parametro **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 : 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)

`segment`: il parametro `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>

`segment`: il parametro `class`

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

Direttiva **assume**

```
ASSUME segm_reg:segm_name {, segm_reg:segm_name}
```

- 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:

```
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
- Può essere usata dovunque all'interno del codice
- Eventuali direttive dello stesso tipo possono specificare nuovamente i registri di segmento in qualunque punto del programma

Direttiva **end**

- Formato:

  END <LABEL>

- 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
              ; definizione dati...
DATI          ENDS

STACK         SEGMENT
              ; definizione stack...
STACK         ENDS

END           BEGIN
```

Un programma sbagliato

```
PROGR1   SEGMENT
         ASSUME   CS:PROGR1, DS:DATI1, SS:STACK
         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
PROGR2   ENDS
DATI1    SEGMENT
         ; definizione dati...
DATI1    ENDS
DATI2    SEGMENT
         PIPPO    DW      ?
         ; definizione dati...
DATI2    ENDS
STACK    SEGMENT
         ; definizione stack...
STACK    ENDS
```

Un programma sbagliato: soluzione 1

```
PROGR1   SEGMENT
         ASSUME   CS:PROGR1, DS:DATI1, SS:STACK, ES:NOTHING
         ; altre istruzioni...
         ; correzione: uso di ES
         ASSUME   ES:DATI2
         MOV      AX, DATI2
         MOV      ES, AX
         MOV      AX, ES:PIPPO
         ; altre istruzioni...
PROGR1   ENDS
PROGR2   SEGMENT
         ASSUME   CS:PROGR2, DS:DATI2, SS:STACK, ES:NOTHING
PROGR2   ENDS
DATI1    SEGMENT
         ; definizione dati...
DATI1    ENDS
DATI2    SEGMENT
         PIPPO    DW      ?
         ; definizione dati...
DATI2    ENDS
STACK    SEGMENT
         ; definizione stack...
STACK    ENDS
```

Un programma sbagliato: soluzione 2

```
PROGR1   SEGMENT
         ASSUME   CS:PROGR1, DS:DATI1, SS:STACK
         ; altre istruzioni...
         ; correzione: cambio temporaneo di DS
         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
PROGR2   ENDS
DATI1    SEGMENT
         ; definizione dati...
DATI1    ENDS
DATI2    SEGMENT
         PIPPO    DW      ?
         ; definizione dati...
DATI2    ENDS
STACK    SEGMENT
         ; definizione stack...
STACK    ENDS
```

Un programma sbagliato: soluzione 3

```
PROGR1   SEGMENT
         ASSUME   CS:PROGR1, DS:DATI1,
                  SS:STACK, ES:DATI2
         ; altre istruzioni...
         ; correzione: altro uso di ES
         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
         ; definizione dati...
DATI1    ENDS
DATI2    SEGMENT
         PIPPO    DW       ?
         ; definizione dati...
DATI2    ENDS
STACK    SEGMENT
         ; definizione stack...
STACK    ENDS
```

Un programma con 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
```

Un programma con CS e DS

```
NAME     PROVA
PROGRAM  SEGMENT
         ASSUME CS:PROGRAM, DS: DATI,
         ASSUME SS: NOTHING, ES: NOTHING
         MOV AX, PROGRAM
         MOV SS, AX
         MOV AX, DATI
         MOV DS, AX
         MOV SP, OFFSET TOP

         MOV AX, ALFA
         ADD AX, BETA
         CALL DIVIDI
         MOV GAMMA, AL
         MOV DELTA, AH
         RET
DIVIDI:  SHR AX,1
         RET
STACK    DW       20 DUP (?)
TOP      LABEL    NEAR
PROGRAM           ENDS
```

```
DATI     SEGMENT
         ALFA     DW       ?
         BETA     DW       ?
         GAMMA    DW       ?
         DELTA    DW       ?
DATI     ENDS
         END
```

Formato delle istruzioni (1)

- Formato delle *istruzioni* Assembly:

  [Label:] Mnemonico [Operandi] [;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

Formato delle istruzioni (2)

- Mnemonico
  - indica l'operazione vera e propria
    - es: operazione di somma => **ADD**
- Operandi
  - la presenza e il numero 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

Linguaggio Assembly x86 (1)

- Case-insensitive
- 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 *token*

Linguaggio Assembly x86 (2)

- *Token*:
  - 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 confuso con una stringa)
    - *costante stringa*
      - qualsiasi combinazione di caratteri fra apici. Es: **`'ALBERO'`**

Linguaggio Assembly x86 (3)

- *Token*:
  - *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 x86 (4)

- 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

Modi di indirizzamento (1)

- Il formato degli operandi dipende dal modo di indirizzamento
- L'8086 ha 7 modi di indirizzamento per i dati:

1.	immediato (dato):	espressione costante
2.	diretto (Effective Address):	nome var. ± espr. costante
	oppure:	nome var. [± espr. costante]
3.	registro:	registro
4.	registro indiretto:	[registro]
5.	registro relativo:	displacement [reg. ± espr. costante]
	oppure:	[reg. ± espr. costante]
6.	basato indicizzato	[reg. base][reg. indice]
7.	rel. basato indic.:	displ. [reg. base ± espr. costante] [reg. indice ± espr. costante]
	oppure:	[reg. base ± espr. costante] [reg. indice ± espr. costante]

Modi di indirizzamento (2)

- Esempi:
  1. immediato: **`10011B, 529, 529D, 'Acqua'`**
  2. diretto: **`CNT-5,[10A0h],CNT, CNT[-5]`**
  3. registro: **`AX`**
  4. registro indiretto: **`[BX]`** (contenuto di BX)
  5. registro relativo **`[AX+10H], 10h[AX], [BX-7D]`**
  6. basato indicizzato **`[BX][SI], [BP][DI]`**
  7. rel. basato indic. **`[BX+2][SI-3], 10A1h[BX][SI], [BP-7H][DI+3H]`**
- Convezione usata nei prossimi lucidi:

SRC	operando sorgente	**SEG**	segmento
REG	registro	**ADDR**	address o indirizzo
DST	operando destinazione		
OPR	operando		
(xxx)	“il contenuto di”		

Istruzioni di trasferimento dati (1)

- istruzioni per il trasferimento dati registro/memoria:
  - **MOV** (move) : **`MOV DST, SRC`**
    copia il contenuto di **SRC** in **DST**
  - **LEA** (load effective address): **`LEA REG, SRC`**
    copia l’indirizzo (offset) di **SRC** in **REG** (in alcuni casi al posto di **LEA** può essere usato l’operatore **OFFSET**)
  - **XCHG** (exchange): **`XCHG OPR1,OPR2`**
    scambia i valori contenuti in **OPR1** e **OPR2**
  - e ancora:
    - **LDS** (load DS con puntatore)
    - **LES** (load ES con puntatore)

Istruzioni di trasferimento dati (2)

- Le istruzioni di trasferimento non affliggono i flags dell'8086
- Se la destinazione è un registro di segmento, la sorgente non può essere un dato immediato
- Per **LEA**:
  - la destinazione non può essere un registro segmento
  - la sorgente non può avere un modo di indirizzamento immediato o registro.
- Per **XCHG:** almeno uno dei due operandi deve essere un registro, ma nessuno dei due deve essere un registro di segmento
- Per **MOV** se uno dei due operandi è specificato col modo immediato, l'altro deve essere un registro.

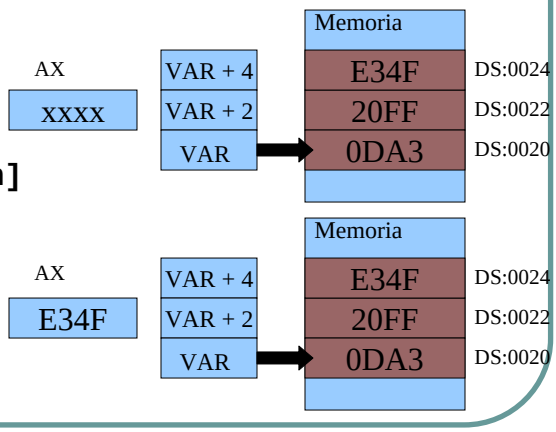
MOV e modi di indirizzamento (1)

- Registro - Immediato

```
MOV BX, 0410h
```

- Registro - Diretto

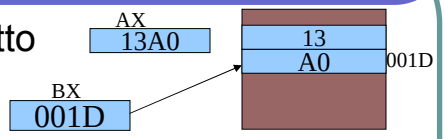
```
MOV AX, VAR[4]
MOV AX, DS:[0024h]
MOV AX, VAR + 4
```



MOV e modi di indirizzamento (2)

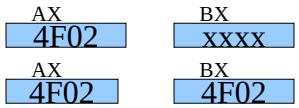
- Registro - Registro Indiretto

```
MOV AX, [BX]
```



- Registro - Registro

```
MOV BX, AX
```

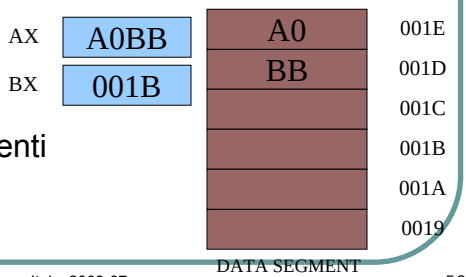


- Registro - Registro Relativo

```
MOV AX, [BX]+2

MOV AX, 4[BP]
MOV AX, [BP+4]
MOV AX, [BP]+4
```

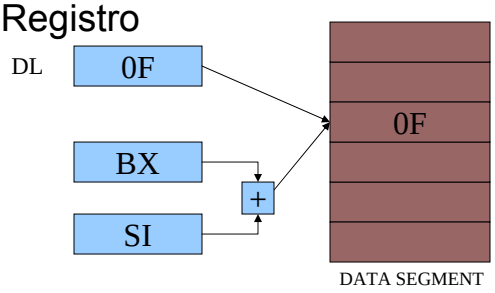
equivalenti



MOV e modi di indirizzamento (3)

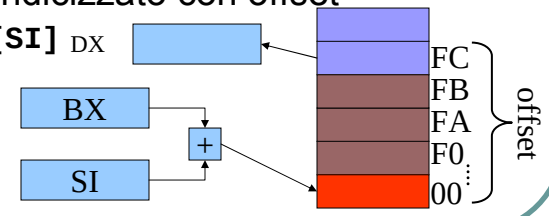
- Basato indicizzato - Registro

```
MOV [BX][SI], DL
```



- Registro - Basato indicizzato con offset

```
MOV DX, 0FCh[BX][SI]
```



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'offset di ARRAY
MOV SI, 0         ; metto 0 in SI
MOV AX, [BX][SI]  ; metto in AX il contenuto
                  ;   dell'indirizzo di memoria ottenuto
                  ;   come somma dei contenuti di BX e SI
```

Dichiarazione dei dati (1)

- Sintassi:

```
<nome_var> db | dw | dd <operandi> [;commento]
```

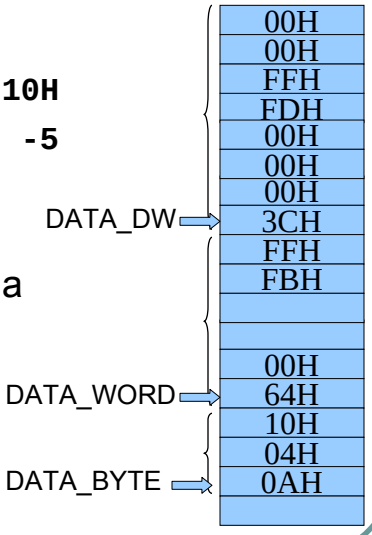
- Lo mnemonico determina la dimensione in byte
  - DB (define byte) ogni operando occupa 1 byte
  - DW (define word) ogni operando occupa 1 word (2 bytes)
  - DD (define double word) occupa 2 word (4 bytes)
- Gli operandi possono essere dei valori o espressioni costanti
- Per riservare lo spazio di memoria si usa "?"

Dichiarazione dei dati (2)

- Esempio:

```
DATA_BYTE    DB   10, 4, 10H
DATA_WORD    DW   100, ?, -5
DATA_DW DD   3*20, 0FFFDH
```

Individuano 3 aree di memoria lunghe rispettivamente:
3 bytes, 6 bytes e 8 bytes



Data Segment

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

Array (1)

- Un *array* è un aggregato di dati tutti dello stesso tipo
- Astrattamente un array può essere 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 (2)

- Come definire un array:

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

Dove:

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

Array (3)

- Esempio 1:

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

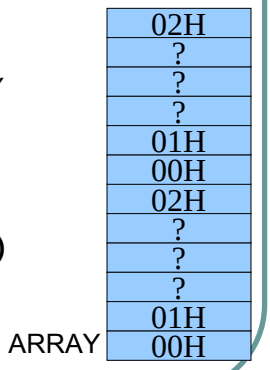
  - V3 è un vettore di memoria di 10 bytes
  - V3 rappresenta l'Indirizzo Base dell'array nel Data Segment
- Esempio 2:

```
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 DUP(0, 1, 3 DUP(?), 2)
```

  - Riserva 12 byte a partire dall'indirizzo ARRAY



Array (4)

- 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 accedere al suo terzo elemento, il codice assembly sarà:

```
[VECT + 2*3]
```

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

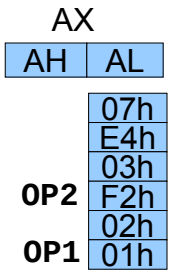
```
MESSAGE DB 'A','L','B','E','R','O‘
```

La variabile **MESSAGE** ha l'indirizzo del byte corrispondente ad 'A'

Assembler e Tipi di dati (1)

- 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:

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



Assembler e Tipi di dati (2)

- Nell'esempio precedente, 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 è formato da 2 byte
  - **OP2** è una word mentre AL è formato da 1 byte

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

Aritmetica Binaria (1)

- Addizione:

  `ADD DST, SRC`
  - destinazione = destinazione + sorgente
- Sottrazione

  `SUB DST, SRC`
  - destinazione = destinazione - sorgente
- Affliggono i flags di condizione

Aritmetica Binaria (2)

- Le funzioni **ADD** e **SUB** settano:
  - il flag di overflow O in presenza di un overflow o underflow con segno
  - il flag di segno S 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 (3)

- 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

Aritmetica Binaria (4)

- Esempio: W = 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
; ...
; definizione dati
X   DW    34
Y   DW    16
Z   DW    20
W   DW    ?
```

Aritmetica Binaria (5)

- Poichè 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, affinchè i risultati siano corretti, occorre usare (anche) riporti e prestiti

Aritmetica Binaria (6)

- Addizione con riporto (add with carry):

  `ADC DST, SRC`

  - destinazione = destinazione + sorgente + carry

- Sottrazione con prestito (subtract with borrow):

  `SBB DST, SRC`

  - destinazione = destinazione - sorgente - carry

- Affliggono i flags di condizione (O, S, Z, C)

Aritmetica Binaria (7)

- 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**
  - Nota: la sottrazione non è commutativa!

Aritmetica Binaria (8)

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
; definizione dati
W11    DW   0123h    ; parte alta del primo addendo
W12    DW   BC62h    ; parte bassa del primo addendo
W21    DW   0012h    ; parte alta del secondo addendo
W22    DW   553Ah    ; parte bassa del secondo addendo
W31    DW   ?        ; parte alta del risultato
W32    DW   ?        ; parte bassa del risultato
```

Aritmetica Binaria (9)

- 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 (10)

- 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 (11)

- 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 (12)

- 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

Aritmetica Binaria (13)

- 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

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 = 2000h
  - Dopo l'esecuzione di IMUL -128×64 = -8192 (in HEX E000h)

    quindi AX = E000h

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

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.

Conversioni di tipo (1)

- **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

Conversioni di tipo (2)

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

```
MOV AX, X     ;metto X in AX
CWD           ;metto in DX l'est. segno di X
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)
```

Conversioni di tipo (3)

- 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 (4)

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

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

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

Conversioni di tipo (5)

- Divisione tra una 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
```

Conversioni di tipo (7)

- Divisione tra una word e una word operando

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

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

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 senza memorizzare il risultato)
- **INC, DEC** e **NEG** alterano i flags: O, S, Z e P
- **NEG** altera anche C
- **CMP** altera opportunamente i flags Z, C, P, S e O

Funzioni Logiche (1)

- **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}
```

Funzioni Logiche (2)

- 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 il 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 flag di condizione.

Funzioni Logiche (3)

- Esempio:
  Se (AL) = 1100 0101 (C5h)
  e (BH) = 0101 1100 (5Ch), l'istruzione:
  AND AL, BH
  fa sì che (AL) diventi 0100 0100 (44h)

- Esempio: Mascheratura
  Se **DATO** = 1001 0001
  e (AL) = 0101 1010, l'istruzione:
  OR AL, DATO
  fa sì che (AL) diventi 1101 1011
  Quindi usare **DATO** come maschera vuol dire settare il 1°, il 5° e l'8° bit dell'altro operando

Funzioni Logiche (4)

- Esempio:
  Se (AX) = 0101 1010
  e **DATO** = 1001 0001, l'istruzione:
  XOR AX, DATO
  fa sì che (AX) = 1100 1011

- Esempio: azzeramento bit con operazione **AND**
  Se (AX) = 0101 1010
  e **DATO** = 1001 0001, allora:
  AND AX, DATO
  fa sì che (AX) = 0001 0000

Funzioni Logiche (5)

- Esempio:
  Se (AX) = 0101 1010
  e **DATO** = 1001 0001, l'istruzione:
  TEST AX, DATO
  genera come risultato 0001 0000 e fa sì che i flag mutino di conseguenza

  - Nel caso in cui il risultato fosse stato 0000 0000, cioè zero, il flag Z sarebbe stato settato
  - Un solo bit uguale a 1 basta per resettare il flag Z

Istruzioni di SHIFT (1)

- Shift logico a sinistra

  SHL OPR, CNT

  - Inserisce in **OPR**, a partire dal bit meno significativo, un numero di zeri pari a **CNT**
  - I bit via via shiftati, che "escono" dalla parte del bit più significativo, vanno a finire in successione nel flag C
- Shift logico a destra

  SHR OPR, CNT

  - Inserisce in **OPR**, a partire dal bit più significativo, un numero di zeri pari a **CNT**
  - I bit via via shiftati, che "escono" dalla parte del bit meno significativo, vanno a finire in successione nel flag C

Istruzioni di SHIFT (2)

- Shift aritmetico a sinistra

  SAL OPR, CNT

  - Uguale a **SHL** (è un sinonimo di **SHL**)
- Shift aritmentico a destra (conservazione del segno)

  SAR OPR, CNT

  - Inserisce in **OPR**, a partire dal bit più significativo, tanti bit con valore pari al segno dell'operando quanti indicati da **CNT**
  - I bit via via shiftati, che "escono" dalla parte del bit meno significativo, vanno a finire in successione nel flag C

Istruzioni di SHIFT (3)

- Il numero di posizioni dello shift è specificato dal contatore **`CNT`** che:
  - se è specificato in *modo immediato* deve valere 1 (ad esempio: **`SHL OPR, 1`**)
  - altrimenti è il contenuto del registro CL
- I flag, 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 nel caso di shift a sx (di fatto si tratta di una moltiplicazione per 2 e può generare overflow) e se il bit più significativo dell'operando fosse 1. Il caso in cui si ha overflow (e quindi) il settaggio di O è quando il nuovo bit più significativo è diverso da quello precedente (depositato nel flag C)

Istruzioni di SHIFT (4)

- Esempio 1:
  - (AH) = 1000 0000 (numero negativo in complemento a 2 pari -128)
  - **`SHL 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 (quindi O = 1) e in effetti lo è, visto che -128 * 2 = -256 (fuori dinamica per una rappresentazione a 8 bit)
- Esempio 2:
  - (AH) = 1100 0000 (numero negativo in complemento a 2 pari a -64)
  - **`SHL AX, 1`** ==> (AX) = 1000 0000 (negativo 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 (5)

Esempi:

- Supponiamo (AH) = 10010011

  SHL AH, 1 => (AH) = 00100110 e C=1

  SHR AH, 1 => (AH) = 01001001 e C=1

  SAR AH, 1 => (AH) = 11001001 e C=1

- Supponiamo (AH) = 10010011 e (CL) = 3

  SHL AH, CL => (AH) = 10011000 e C=0

  SHR AH, CL => (AH) = 00010010 e C=0

  SAR AH, CL => (AH) = 11110010 e C=0

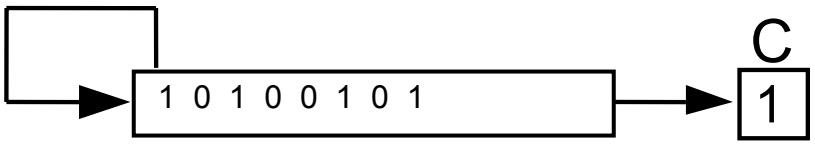
Istruzioni di SHIFT (6)

Esempio: una serie di **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

Istruzioni di SHIFT (7)

Esempio: una serie di **`SAR AL, 1`**

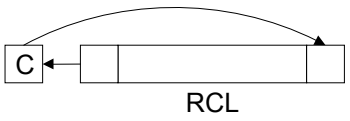


1 1 0 1 0 0 1 0	1
1 1 1 0 1 0 0 1	0
1 1 1 1 0 1 0 0	1
1 1 1 1 1 0 1 0	0

Istruzioni di ROTATE (1)

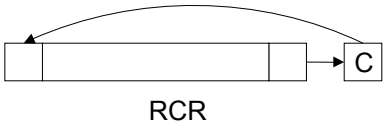
- Rotate: simile allo shift, ma i bit "rientrano" dalla parte opposta
- **`RCL`** (Rotate through Carry Left)

```
RCL OPR, CNT
```



- **`RCR`** (Rotate through Carry Right)

```
RCR OPR, CNT
```

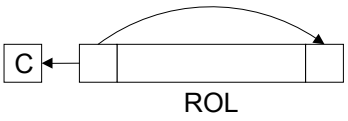


- Solo C è affetto dall'istruzione
- **`CNT`** è 1 (nel caso immediato) o CL
- **`OPR`** può essere un dato con indirizzamento qualsiasi eccetto l'immediato

Istruzioni di ROTATE (2)

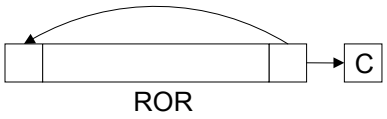
- **ROL** (ROtate Left)

```
ROL OPR, CNT
```



- **ROR** (ROtate Right)

```
ROR OPR, CNT
```



- Solo C è affetto dall'istruzione
- C contiene una replica del bit estratto
- **CNT** è 1 (nel caso immediato) o CL
- **OPR** può essere un indirizzamento qualsiasi eccetto l'immediato

Istruzioni di ROTATE (3)

- 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
…
```

Definizione di Costanti (1)

- 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
  - L'assemblatore non alloca memoria per le costanti (a differenza di quanto accade per le variabili): si limita a sostituire il nome simbolico della costante con il valore che le è stato assegnato => Una costante "non ha 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
```

Definizione di Costanti (2)

- Esempi:

```
K EQU 1024
```

Ogni occorrenza di **K** all'interno del codice equivale alla presenza del numero 1024. Quindi il codice:

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

è del tutto equivalente a:

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

La direttiva **LABEL (1)**

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

```
nome LABEL tipo
```

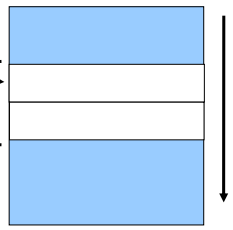
dove tipo può essere:

- per le etichette vere e proprie: NEAR, FAR (salti)
  - NEAR: label associata solo all'offset (default)
  - FAR: label associata a segmento e offset
- per le variabili: BYTE, WORD o DWORD

La direttiva **LABEL** (2)

- Label usata come variabile:

```
BETA LABEL BYTE
ALFA DW ?
```



- **BETA** punta alla parte bassa di **ALFA** (locazione di memoria riservata immediatamente dopo)
- 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

La direttiva **LABEL** (3)

- Label usata come etichetta:

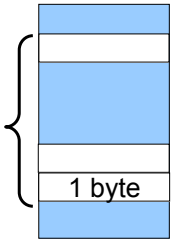
```
NO_VAL    LABEL FAR
MOV       AX, 05Fh
```

NO_VAL punta all'indirizzo dell'istruzione che la segue

- Label usata con gli array

```
BYTE_ARRAY LABEL BYTE
WORD_ARRAY DW 50DUP(?)
```

riserva 100 locazioni di memoria di un byte accessibili byte a byte se si usa **BYTE_ARRAY**, word a word se si usa **WORD_ARRAY**



Operatore **OFFSET (1)**

- 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

Operatore **OFFSET (2)**

- L'operatore **OFFSET** non 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 (1)**

- Formato:

```
TYPE <variabile>
LENGTH <variabile>
SIZE <variabile>
```

  - **TYPE** restituisce il numero di byte dell'operando; applicato ad una label: -1 se **NEAR**, -2 se **FAR**
  - **LENGTH** 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 (2)**

- **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, LENGHT EXP   ;(AX) = 05H
        MOV BX, TYPE EXP     ;(BX) = 02H
        MOV CX, SIZE EXP     ;(CX) = 0AH
        MOV DX, LENGHT EXP2  ;(DX) = 01H
```

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
```

Operatore PTR (1)

- Formato:

  tipo PTR <variabile>±<espressione costante>

  - **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 (2)

- Esempio 1:

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

Discordanza sui tipi e pertanto non assemblabile

- 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
…
```

Operatore PTR (3)

- Esempio 2:

```
ARRAY DB 2 DUP(1)
…
MOV   AX, ARRAY              ;Errore!!
MOV   AX, WORD PTR ARRAY     ;OK
…
```

Istruzioni di salto (1)

- 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*

Istruzioni di salto (2)

- 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

Modi di indirizzamento per i salti (1)

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

Modi di indirizzamento per i salti (2)

- Indirizzamento per salti intrasegmento:
  - *diretto* (label)**:** l'EA (Effective Address) cui saltare si ottiene come somma di (IP) e di uno scostamento a 8 bit (*salto corto o "short"*) o a 16 bit (*salto vicino o "near"*)
    - si usa sia per il salto condizionato che per quello incondizionato. Per il primo salto però vale solo lo scostamento corto
  - *indiretto***:** 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)
    - si usa solo nel salto incondizionato
- Indirizzamento per salti intersegmento (solo salti incondizionati):
  - *diretto* (label)**:** vengono sostituiti (IP) e (CS) con un offset ed un segmento forniti direttamente dall'istruzione di salto (attraverso una label FAR)
  - *indiretto***:** si sostituiscono (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 (1)

- 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
- Direzione del salto:
  - Scostamento >0: salto in avanti
  - Scostamento <0: salto indietro

Salti Condizionati (2)

- 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

Salti Condizionati (3)

- Esempio (in cui si ipotizza che ogni istruzione venga codificata in 2 byte):

```
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).

JXX (senza segno)

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**

Paolo Nesi, Univ. Firenze, Italy, 2003-07 594

JXX (con segno)

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**

Paolo Nesi, Univ. Firenze, Italy, 2003-07 595

JXX (aritmetici)

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**

Salti Incondizionati (1)

- 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

Salti Incondizionati (2)

- Intrasegmento
  - diretto (label) - salto corto

```
JMP SHORT OPR ;(IP)=(IP)+scost. a 8 bit
              ; cioè fino a 128 byte
```

  - diretto (label) - salto vicino

```
JMP NEAR OPR   ;(IP)=(IP)+scost. a 16 bit
               ; cioè fino a 32000 byte
```

  - indiretto

```
JMP OPR        ;(IP)=(EA) con EA
               ;determinato da OPR
               ;(no ind. immediato)
```

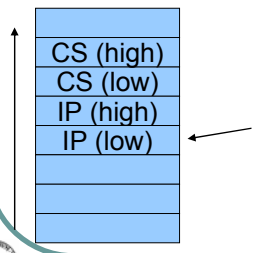
Salti Incondizionati (3)

- Intersegmento
  - diretto (label) - salto lontano

```
JMP FAR OPR   ;(IP)=EA di OPR
              ;(CS)=indirizzo segmento di OPR
```

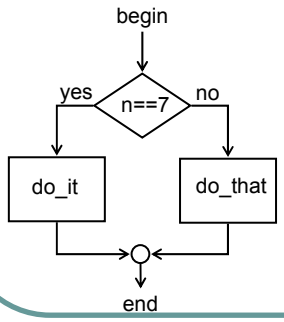
  - indiretto

```
JMP OPR       ;(IP)=(EA) con EA determinato
              ;da OPR e (CS)=(EA+2)
              ;(no ind. registro e imm.)
```



Implementare un IF-ELSE

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

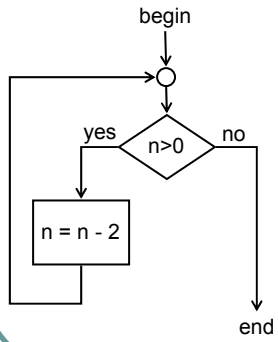



```
        cmp n,'7'
        jne else_
        ;do_it…
        jmp short endif
else_:
        ;do_that…
endif:
```

Paolo Nesi, Univ. Firenze, Italy, 2003-07 600

Implementare un WHILE

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




```
while_:
        cmp  n,0
        jle  end_while

        sub  n,2
        jmp  while_
end_while:
```

Paolo Nesi, Univ. Firenze, Italy, 2003-07 601

Valutazioni di espressioni logiche: AND

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

```
        cmp n,'A'
        jl  no_go
        mov ax,w
        cmp ax,x
        jne no_go

        ;whatever…
no_go:
```

Valutazioni di espressioni logiche: OR

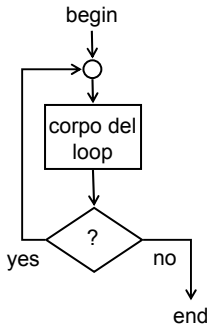
```
char n,k;
unsigned int w;
…
if (n<>k || w<=10)
  whatever();
```

```
        mov ah,n
        cmp ah,k
        jne then_
        cmp w,10
        ja end_if
then_:
        ;whatever…
end_if:
```

Cicli

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

```
      …
        MOV CX, N  ;uso CX come contatore
INIZIO:
      ;"corpo" del loop
FINE:
        DEC CX     ;decremento 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

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
```

Istruzione **LOOP (2)**

```
for (x=9; x>0; x--)
  n = n + x;
```

```
            MOV CX,9
TOP_LOOP:   ADD N,CX
            LOOP TOP_LOOP
```

Istruzioni LOOP (1)

- 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 (2)

- 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

Istruzioni LOOP (3)

- **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 **STR**.

```
        ...
        MOV CX, L           ;inizializzo il contatore
        MOV SI, -1          ;inizializzo l'indice
        MOV AL, 20h         ;metto in AL il codice da trovare
        MOV BX, OFFSET STR
START:  INC SI              ;si incrementa l'indice
        MOV DX, [BX][SI]    ;metto in DX il carattere n-esimo
        CMP DX, 20h         ;confronta lo spazio
                            ;con il carattere puntato
        LOOPNE START        ;se non sono uguali (e CX non è
                            ;arrivato a zero) cerco ancora
        ...
```

Istruzioni LOOP (4)

- Si può saltare all'interno di un loop
- Sono possibili loop annidati
- In caso di loop annidati, ci può essere un conflitto derivante dalla condivisione dello stesso contatore CX. Per cui, quando si accede ai loop via via più interni, è necessario salvare il contenuto di CX: in questo modo si può ripristinare il valore del contatore quando si ritorna ai loop più esterni

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

Nuovo Ordinamento

Parte 10: Procedure, Stack ed Interruzioni

Prof. Paolo Nesi
Ing. Paolo Vaccari
http://www.dsi.unifi.it/~nesi

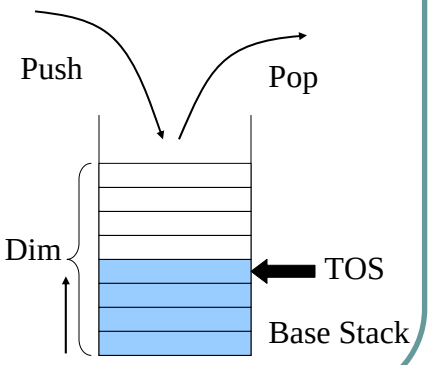
2007

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

Corpo del Loop #1
Corpo del Loop #2
Corpo del Loop #3

La memorizzazione di (CX) è facilitata da una struttura LIFO (Last In - First Out) detta STACK



Lo Stack nell'8086

- Implementazione basata su Array
- Un registro dedicato (SP) tiene traccia del TOS (Top Of the Stack)
- 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]
```

Perché avere uno Stack?

- L'8086 ha istruzioni per gestirlo
- Serve al processore in occasione di interrupt (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 (1)

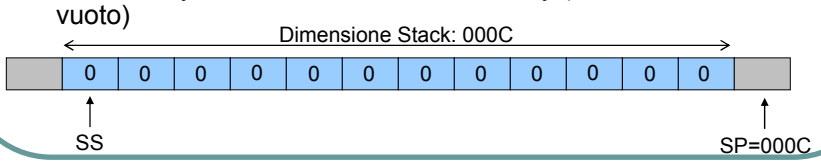
- Tutti i programmi eseguibili devono definire un segmento per lo stack
  - Un array di bytes accessibili attraverso il registro SS e un offset
- SS punta all'inizio dell'area di memoria
- SP è l'offset del TOS (top of the stack)
- Il loader del S.O. inizializza questi registri prima che l'esecuzione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

Stack (2)

- Definizione di uno stack segment:

```
STACK_SEG       SEGMENT     STACK
The_Stack       DB      12      dup  (0)
STACK_SEG       ENDS
```

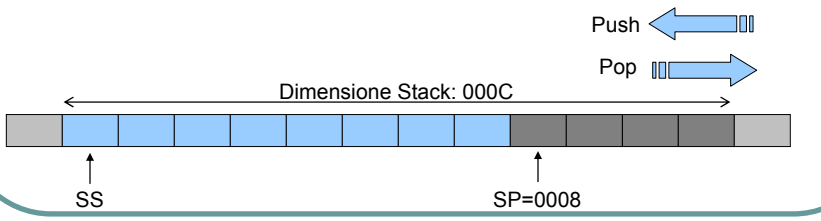
  - Definisce lo stack duplicando 12 byte inizializzati a 0
- Al caricamento…
  - SS è fissato all'indirizzo di segmento contente l'array (di solito **`The_Stack`** parte all'offset 0)
  - SP è fissato all'offset dato da:
    `The_Stack + <dimensione stack>`
    che è un byte oltre la fine dello stack array (condizione di stack vuoto)



Paolo Nesi, Univ. Firenze, Italy, 2003-07 616

Come funziona lo Stack

- Lo stack cresce all'indietro nella memoria in direzione dell'inizio del segmento di stack
- **PUSH** decremena lo stack pointer
  POP incrementa lo stack pointer

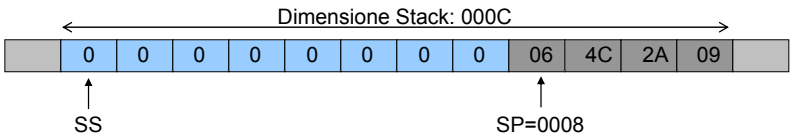


Paolo Nesi, Univ. Firenze, Italy, 2003-07 617

L'istruzione **PUSH** (1)

- **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
- **PUSHF (PUSHFD)**
  - copia il FLAGS register nello stack

L'istruzione **PUSH** (2)

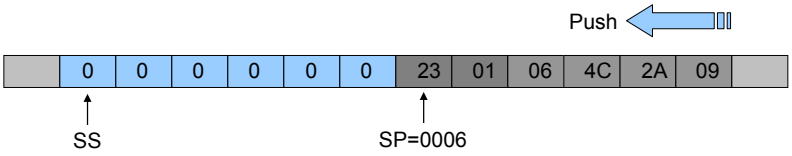


- Esempio:

  se (AX) = 0123, dopo l'esecuzione dell'istruzione

  PUSH AX

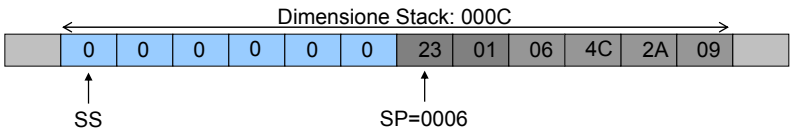
  il contenuto dello stack diventa:



L'istruzione **POP** (1)

- **POP DST**
  - **DST** è un registro a 16 bit generico o di segmento (escluso CS), oppure l'indirizzo di una word o double word
- **POPF (POPFD)**
  - Copia il top of stack nel FLAGS register

L'istruzione **POP** (2)

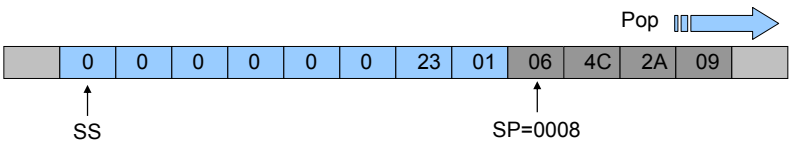


- Esempio:

  dopo l'esecuzione dell'istruzione

  POP ES

  (ES) = 0123 e il contenuto dello stack diventa:



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 (in complemento a 2)
  - Si ha Underflow se SP diventa più grande del suo valore iniziale

Procedure (1)

- 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
- Alla fine della procedura un altro salto riporta all'istruzione successiva alla chiamata

Procedure (2)

```
proc_nome PROC tipo
   …      ;corpo della procedura
   RET    ;ritorno al punto di chiamata
proc_nome ENDP
```

- **tipo** vale NEAR o FAR
  - 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
- Le procedure possono avere uno o più **RET**

CALL e RET (near)

- Chiamata di una procedura NEAR

  CALL proc_name
  - effettua il push dell'IP nello stack
  - l'indirizzo di **proc_nome** è copiato nell'IP
- Ritorno da una procedura NEAR

  RET [n]
  - effettua il pop del top of stack in IP
  - aggiunge **n** a SP (opzionale)

CALL e RET (far)

- Chiamata di una procedura FAR

  `CALL FAR proc_name`

  - effettua il push di CS e IP nello stack
  - l'indirizzo far di **`proc_name`** è copiato in CS:IP
- Ritorno da una procedura FAR

  `RET [n]`

  - effettua il pop di top of stack in IP e poi in CS
  - aggiunge **n** a SP (opzionale)

Interrupt (1)

- 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 processore)
- Esistono due tipi di interrupt:

1. *Interrupt Hardware*
   - Generati dall'esterno (I/O)
   - Consentono di gestire eventi asincroni
   - Possono essere mascherabili o non mascherabili
2. *Interrupt Software*
   - Generati durante l'esecuzione del programma
   - Direttamente mediante istruzioni esplicite (per esempio: **`INT`**)
   - Indirettamente, nel caso si verifichino particolari eventi all'interno del processore (es: divisione per zero)

Interrupt (2)

- Le locazioni di memoria da 0h a 3FFh (1023) contengono l'*Interrupt Vector Table*, formata da 256 elementi. Ogni elemento 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 5 elementi della tabella sono dedicati a particolari tipi di interrupt predefiniti
- 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`**

Interrupt (3)

- Gli interrupt (quelli *mascherabili*) possono essere disabilitati resettando l'Interrupt Flag (I) con l'istruzione **`CLI`** *(clear interrupt)*
- Gli interrupt possono essere abilitati con l'istruzione **`STI`** *(set interrupt)*
- Quando avviene un interrupt, la CPU completa l'istruzione corrente, quindi:

1. Disabilita gli interrupt mascherabili (**`CLI`**), per evitare 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 nell'elemento N dell'Interrupt Vector Table (cioè all'indirizzo di memoria 4 * N, dove N è il numero di interrupt)
4. Esegue la routine di servizio associata all'interrupt (ISR)
5. Al termine della routine esegue un'istruzione **`IRET`** per ripristinare IP, CS e il flags register con i valori presi dallo stack (**`POP`**), ripristinando così lo stato che la CPU aveva prima dell'arrivo dell'interruzione

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

Interrupt 21h (1)

- In MS-DOS, Int 21h fornisce varie funzionalità, fra cui 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

Interrupt 21h (2)

- Si sceglie la funzione desiderata ponendo un opportuno valore nel registro AH. In base alla funzione scelta si dovranno utilizzare altri registri per passare dei parametri
- Terminazione del programma (4Ch):

```
MOV AH, 4Ch
INT 21h
```

- 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 il carattere sullo schermo
```

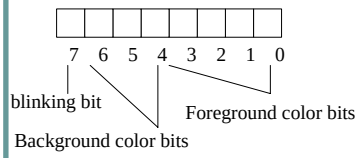
Interrupt 10h (1)

- Clear screen: INT 10h Service 06h

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)

Interrupt 10h (2)

- Color : BH



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

Interrupt 10h (3)

- La posizione del cursore determina dove visualizzare il prossimo carattere
- INT 10h Service 02h permette di impostare la posizione del cursore
- INT 10h Service 03h permette di conoscere la posizione del cursore
- BH definisce il numero di pagina
- DH e DL definiscono la riga (y) e la colonna (x) per la posizione

Interrupt 10h (4)

- L'Interrupt 10h con servizio 08h permette di leggere il carattere su cui è posizionato il cursore:

```
MOV  AH, 08h
MOV  BH, 00h
INT  10h
```

- Il carattere potrà essere letto nel registro AL in rappresentazione ASCII.

Comunicazione con dispositivi I/O (1)

- L'istruzione **IN** 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
```

Comunicazione con dispositivi I/O (2)

- L'istruzione OUT 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
```

Comunicazione con dispositivi I/O (3)

Esempio :

```
PORTA EQU 0FFH      ; indirizzo della periferica
MOV AL, 01h
OUT PORTA, AL       ; attiva la periferica
… ; comunicazione con la periferica
MOV AL, 0h
OUT PORTA, AL       ; disattiva la periferica
```

- Si ipotizza di poter attivare o disattivare la periferica all'indirizzo 0FFh inviandole un bit
- Inviando un bit per mezzo della OUT si attivano sia la decodifica dell'indirizzo che il segnale IOWR e la loro contemporaneità causa l'accesso alla periferica

Hello world! (1)

```
; file hello1.asm
c_seg    SEGMENT
         ASSUME cs:c_seg, ds:d_seg, ss:s_seg
inizio:  mov ax, d_seg              ; inizializzo il DS
         mov ds, ax
         mov dx, OFFSET mesg
         mov ah, 09h                ; stampo la stringa puntata da DS:DX
         int 21h                    ; chiamo l'interrupt DOS
         mov ah, 4Ch                ; termino il programma
         int 21h                    ; chiamo l'interrupt DOS
c_seg    ENDS
d_seg    SEGMENT para public 'data'
mesg     db "Hello world!",13,10,'$' ; 13=CR, 10=LF, $=terminator
d_seg    ENDS
s_seg    SEGMENT para stack 'stack'
                                    ; vuoto!
s_seg    ENDS
         END inizio
```

- Assembling, linking and executing:

```
> tasm hello1.asm
> tlink hello1.obj
> hello1.exe
Hello world!
```

Hello world! (2)

```
; file hello2.asm
.MODEL  small                       ; modello di memoria da usare
.STACK  10                          ; dimensiona lo stack
.DATA                               ; inizio del segmento dati
mesg    db "Hello world!",13,10,'$' ; 13=CR,10=LF,$=terminatore
.CODE                               ; inizio del segmento codice
inizio:mov ax, SEG mesg             ; ax = indirizzo Seg. Dati
       mov ds, ax
       mov dx, OFFSET mesg          ; dx = offset Seg. Dati
       mov ah, 09h                  ; stampo la stringa in DS:DX
       int 21h                      ; chiamo l'interrupt DOS
       mov ah, 4Ch                  ; termino il programma
       int 21h                      ; chiamo l'interrupt DOS
       END inizio
```

- Assembling, linking and executing:

```
> tasm hello2.asm
> tlink hello2.obj
> hello2.exe
Hello world!
```

Hello world (3)

```
; file hello3.asm
c_seg      SEGMENT
           ASSUME cs:c_seg, ds:d_seg, ss:s_seg
inizio:    mov ax, d_seg                  ; inizializzo il DS
           mov ds, ax
           mov ax, s_seg                  ; inizializzo il SS
           mov ss, ax
           call hello
           call hello
           mov ah, 4Ch                    ; termino il programma
           int 21h                        ; chiamo l'interrupt DOS
hello      PROC near
           mov dx, OFFSET mesg
           mov ah, 09h                    ; stampo la stringa puntata da DS:DX
           int 21h                        ; chiamo l'interrupt DOS
           ret
hello      ENDP
c_seg      ENDS
d_seg      SEGMENT para public 'data'
mesg       db "Hello world!",13,10,'$'    ; 13=CR, 10=LF, $=terminatore
d_seg      ENDS
s_seg      SEGMENT para stack 'stack'
           db 20 dup (0FFh)               ; 20 byte per lo stack
s_seg      ENDS
           END inizio
```

- Assembling, linking and executing:
  > tasm hello3.asm
  > tlink hello3.obj
  > hello3.exe
  Hello world!
  Hello world!

Calcolatori Elettronici

CDL in Ingegneria Elettronica

Facoltà di Ingegneria,
Università degli Studi di Firenze

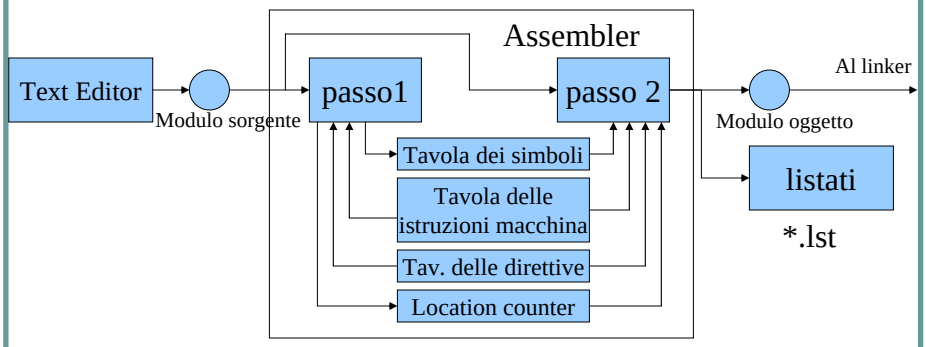
Nuovo Ordinamento

Parte 11: Esempi di Programmazione

Prof. Paolo Nesi
Ing. Paolo Vaccari
http://www.dsi.unifi.it/~nesi

2007

Assemblatore a 2 passi



Paolo Nesi, Univ. Firenze, Italy, 2003-07 644

Assemblatore Passo 1

- Fornire le locazioni degli identificatori (variabili e label) per costruire la Tavola dei Simboli
- 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 quantità 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

Paolo Nesi, Univ. Firenze, Italy, 2003-07 645

Listato e Tavola dei Simboli

> tasm /l/zi hello3.asm → hello3.lst

```
1   0000    c_seg       SEGMENT
2           ASSUME cs:c_seg,ds:d_seg,ss:s_seg
3
4   0000    inizio:     mov ax, d_seg
5   0003                mov ds, ax
6   0005                mov ax, s_seg
7   0008                mov ss, ax
8
9   000A                call hello
10  000D                call hello
11  0010                call hello
12
13  0013                mov ah, 4Ch
14  0015                int 21h
15
16  0017    hello       PROC near
17  0017                mov dx, OFFSET mesg
18  001A                mov ah, 09h
19
20  001C                int 21h
21  001E                ret
22  001F    hello       ENDP
23
24  001F    c_seg       ENDS
25
```

```
26  0000    d_seg       SEGMENT para public 'data'
27  0000    mesg        db "Hello world!",13,10,'$'
28
29
30  000F    d_seg       ENDS
31
32  0000    s_seg       SEGMENT para stack 'stack'
33  0000                db 20 dup (0FFh)
34  0014    s_seg       ENDS
35                      END inizio


Symbol Name Type        Value

HELLO       Near        C_SEG:0017
INIZIO      Near        C_SEG:0000
MESG        Byte        D_SEG:0000


Segments    Bit Size  Align   Combine
            Class

C_SEG       16  001F  Para    none
D_SEG       16  000F  Para    Public  DATA
S_SEG       16  0014  Para    Stack STACK
```

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

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 offset da associare

Osservazioni (1)

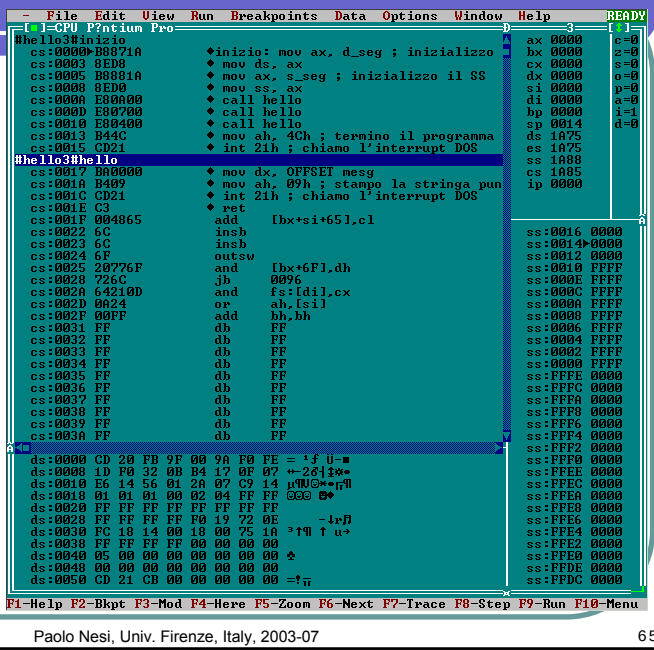
- Quando l'assemblatore al passo 1 incontra in un operando una variabile o label:
  - questa può già trovarsi nella Tabella dei Simboli (BACKWARD REFERENCE)
  - Se la variabile o la label non compare nella Tavola dei Simboli vuol dire che ancora non è stata incontrata una direttiva che l'ha definita 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)

Osservazioni (2)

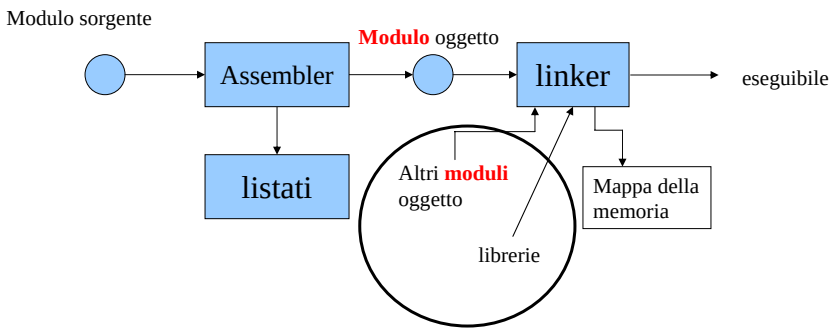
- L'inserimento di **NOP** rallenta l'esecuzione
- Per ovviare al problema delle forward reference una soluzione è esplicitare il tipo della variabile o label usando l'operatore **PTR:**
  - **BYTE PTR, WORD PTR o DWORD PTR** per le variabili
  - **NEAR PTR o FAR PTR** per le label
- Si ha forward reference 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 (per esempio: **ES:VAR**) oppure settare momentaneamente il DS al segmento per quella variabile e poi ripristinare il precedente valore di DS

Il debugger

- Un debugger è molto utile per comprendere meglio i passi compiuti dall'assembler e la logica di esecuzione del programma
- Nella figura, la finestra di visualizzazione "CPU" del Borland Turbo Debugger (td), durante il debug dell'esempio "Hello world!#3":
  - > td hello3.exe
  - le tre finestre principali "fotografano" lo stato dei tre segmenti del programma
  - la finestra più piccola riporta il contenuto dei registri del processore e dei flag di stato
  - utilizzando il tasto F7 (funzione "Step into") è possibile eseguire il programma un'istruzione alla volta e monitorare le variazioni del contenuto di memoria e registri



Il linker



Programmazione modulare

Programmazione modulare (1)

- Per i sottoprogrammi:
  - **`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`**

Programmazione modulare (2)

- 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**

Esempio: modulo 1

```
PUBLIC          V1, V2, V3
EXTRN           PROC_A:FAR

DATA_SEG1       SEGMENT
V1              DW ?
V2              DW ?
V3              DW ?
DATA_SEG1       ENDS

STACK_SEG SEGMENT
                DW 30 DUP(?)
TOS             LABEL WORD
STACK_SEG ENDS
```

```
CODE_SEG        SEGMENT
                ASSUME CS:CODE_SEG,
                DS:DATA_SEG1,
                SS:STACK_SEG
START:          MOV AX, DATA_SEG1
                MOV DS, AX
                MOV AX, STACK_SEG1
                MOV SS, AX
                MOV SP, OFFSET TOS
                …
                MOV V1, AX
                MOV V2, BX
                CALL FAR PTR PROC_A
                …
CODE_SEG        ENDS
                END START
```

Esempio: 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
```

Calcolatori Elettronici

Fine del corso

Prof. Paolo Nesi
http://www.dsi.unifi.it/~nesi
nesi@dsi.unifi.it
AA 2006-2007