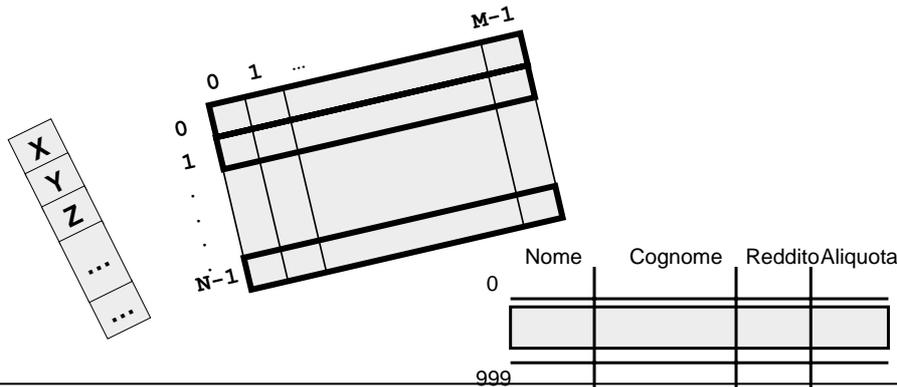


# Il linguaggio C

## Tipi strutturati



Fondamenti di Informatica L- A

## Tipi di dato strutturati

I dati strutturati (o strutture di dati) sono ottenuti mediante composizione di altri dati (di tipo semplice, oppure strutturato).

### Tipi strutturati in C:

- **vettori** (o *array*)
- **record** (*struct*)
- [record varianti (*union*)]

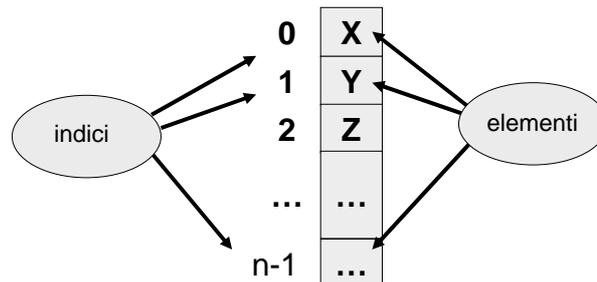
Fondamenti di Informatica L- A

# Vettori (o *array*)

Un **vettore** (o **array**) è un insieme ordinato di elementi tutti dello stesso tipo.

## Caratteristiche del vettore:

- omogeneità
- ordinamento, ottenuto mediante dei valori interi (**indici**) che consentono di accedere ad ogni elemento della struttura.



Fondamenti di Informatica L- A

# Definizione di vettori in C

Nel linguaggio C per definire vettori, si usa il **costruttore di tipo** `[]`.

## Sintassi:

```
<id-tipo> <id-variabile> [ <dimensione> ] ;
```

## dove:

- **<id-tipo>** è l'identificatore di tipo degli elementi componenti,
- **<dimensione>** è una costante intera che rappresenta il numero degli elementi componenti,
- **<id-variabile>** è l'identificatore della variabile strutturata (il vettore) così definita

## Esempio:

```
int V[10]; /* vettore di dieci elementi interi */
```

**NB:** La dimensione (numero di elementi del vettore) deve essere una **costante intera**, nota al momento della dichiarazione.

```
int N;  
char V[N]; /* def. sbagliata!!! */
```

Fondamenti di Informatica L- A

# Vettori in C

```
int v[25]; /*def. di un vettore di 25 elementi interi*/
```

## Indici:

- ad ogni elemento è associato univocamente un **indice**.
- è possibile riferire una singola componente specificando l'indice  $i$  corrispondente, utilizzando la notazione  $V[i]$ .
- L'indice deve essere di tipo enumerabile (ad esempio: **int**, **char**).
- se  $N$  è la **dimensione** del vettore (numero degli elementi), il dominio degli indici è l'intervallo

**[0,N-1]**

## Esempio:

```
float A[3], i; /* A è un vettore di _____ */
A[0]=22.05; /*ass. di un valore al primo elemento */
A[2]=17.9; /*ass. di un valore al _____ elemento */
A[1]=0; /*ass. di un valore al _____ elemento */
```

A

---

Fondamenti di Informatica L- A

# Vettori in C

## Operatori:

In C **non esistono operatori specifici** per i vettori.

Per operare sui vettori è necessario operare singolarmente sugli elementi componenti (coerentemente con il tipo ad essi associato).

## Pertanto:

→ **non è possibile l'assegnamento diretto** tra vettori:

```
int v[10], w[10];
...
v=w; /* è scorretto! */
```

→ **non è possibile leggere (o scrivere)** un intero vettore (fanno eccezione, come vedremo, le stringhe); occorre leggere/scrivere le sue componenti;

## Esempio: lettura di un vettore:

```
int v[100];
int i;

for(i=0; i<100; i++)
{
    printf("valore %d-simo? ", i);
    scanf("%d", &v[i]);
}
```

---

Fondamenti di Informatica L- A

## Gestione degli elementi di un vettore

Le singole componenti di un vettore possono essere elaborate con gli operatori del tipo ad esse associato.

**Esempio:** agli elementi di un vettore di interi è possibile applicare tutti gli operatori definiti per gli interi:

```
int A[10], n=7 i=2;
A[i] =n%i;          /* A[ ]= */
A[9]=A[i++]+7;     /* A[9]= i= */
scanf("%d" (&A[i])); /*ass. ad A[ ] /
```

---

Fondamenti di Informatica L- A

## typedef ... .. [ ]

In C è possibile utilizzare `typedef ... .. [ ]` per dichiarare tipi di dato non primitivi rappresentati da vettori.

**Sintassi della dichiarazione:**

```
typedef <tipo-componente> <tipo-vettore> [<dim>];
```

dove:

- `<tipo-componente>` è l'identificatore di **tipo di ogni singola componente**
- `<tipo-vettore>` è l'identificatore che si attribuisce al **nuovo tipo**
- `<dim>` è il **numero di elementi** che costituiscono il vettore (deve essere una costante).

**Esempio:**

```
typedef int Vettore[30];
Vettore v1,v2;. /* v1 e v2 sono variabili di tipo
                Vettore; ognuno rappresenta un vettore
                di 30 elementi interi. */
```

→ **v1 e v2 possono essere utilizzati come vettori di interi.**

---

Fondamenti di Informatica L- A

# Vettori in sintesi

Riassumendo:

- **Definizione di variabili di tipo vettore:**

```
<tipo-componente> <nome-variabile>[<dim>];
```

- **Dichiarazione di tipi non primitivi rappresentati da vettori:**

```
typedef <tipo-componente> <nome-tipo> [<dim>];
```

**Caratteristiche:**

- <dim> è una costante enumerabile.
- <tipo-componente> è un qualsiasi tipo, semplice o strutturato.
- il vettore è una sequenza di dimensione fissata <dim> di componenti dello stesso tipo <tipo-componente>.
- la singola componente i-esima di un vettore V è individuata dall'indice i-esimo, secondo la notazione  $V[i]$ .
- L'intervallo di variazione degli indici è [ 0 , ... , dim-1 ]
- sui singoli elementi è possibile operare secondo le modalità previste dal tipo <tipo-componente>.

---

Fondamenti di Informatica L- A

# Inizializzazione di un vettore

Come attribuire un valore iniziale ad ogni elemento di un vettore?

2 possibilità. 1) **Mediante un ciclo:**

Per attribuire un valore iniziale agli elementi di un vettore, si può attuare con una sequenza di assegnamenti alle N componenti del vettore.

**Esempio:**

```
#define N 30
typedef int vettore [N];
vettore v;
int i;
...
for( i=0; i<N; i++ )
    v[i]=0;
```

**#define:** La **#define** è una direttiva del *preprocessore* C che serve ad associare un identificatore (nell'esempio N) ad una costante (nell'esempio, 30). Rende il programma più facilmente modificabile.

---

Fondamenti di Informatica L- A

# Inizializzazione di un vettore

Come attribuire un valore iniziale ad ogni elemento di un vettore?

2) In alternativa al ciclo, è possibile l'inizializzazione in fase di definizione:

**Esempio:**

```
int v[10] = {1,2,3,4,5,6,7,8,9,10};  
  
/* v[0] = 1; v[1]=2;...v[9]=10; */
```

**Addirittura è possibile fare:**

```
int v[]={1,2,3,4,5,6,7,8,9,10};
```

➔ La dimensione è determinata sulla base dell'inizializzazione.

---

Fondamenti di Informatica L- A

## Esempio

**Problema:** (somma di due vettori)

Si realizzi un programma che, dati da standard input gli elementi di due vettori A e B, entrambi di 10 interi, calcoli e stampi gli elementi del vettore C (ancora di 10 interi), ottenuto come somma di A e B.

- Dichiariamo il nuovo tipo di dato **vettint**:

```
_____ ;
```

- Definiamo i due vettori dati A e B, e il vettore C risultante dalla somma :

```
_____
```

---

Fondamenti di Informatica L- A

```

#include <stdio.h>
typedef int vettint[10];

main()
{ vettint A, B, C;
  int i;
  for(i=0; i<10; i++) /* lettura del vettore A */
  { printf("valore di A[%d] ? ", i);
    scanf("%d", );
  }
  for(i=0; i<10; i++) /* lettura del vettore B */
  { printf("valore di B[%d] ? ", i);
    scanf("%d", );
  }

  /* calcolo del risultato */

  for(i=0; i<10; i++) /* stampa del risultato C */
  printf("C[%d]=%d\n", i, );
}

```

---

Fondamenti di Informatica L- A

## Esempio

(calcolo del numero medio di iscritti per classe)

Si consideri una scuola media costituita da 4 sezioni (da 'A' a 'D'), ognuna costituita da tre classi ("livello" 1, 2, 3).

Dati gli iscritti ad ogni classe di ogni sezione, si vuole calcolare il numero medio di studenti per classe relativo ad ogni sezione.

Definiamo un nuovo tipo che rappresenta il **livello**:

```
/* un elemento per ogni sezione */
```

```
typedef
livello prime, seconde, terze;
```

s='A', s-'A' = 0	n0
s='B', s-'A' = 1	n1
s='C', s-'A' = 2	n2
s='D', s-'A' = 3	n3

**NB:** L'indice di un vettore deve essere di tipo **enumerabile** → può essere un **char**:

```
typedef char sezione;
sezione s; // s = 'A', 'B', ...
```

```
/*indice per accedere ai vettori delle classi */
```

---

Fondamenti di Informatica L- A

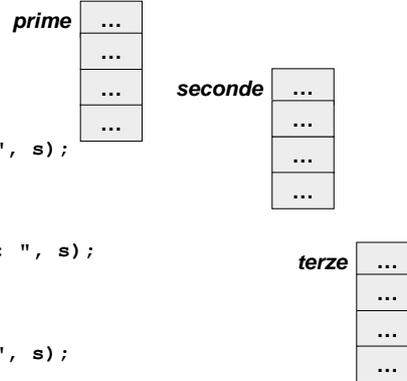
indici

elementi

```

#include <stdio.h>
typedef int livello[4]; /*un elemento per ogni sezione */
typedef char sezione;
main()
{
    livello prime, seconde, terze;
    sezione s;
    float media=0;
    for(s='A'; s<='D'; s++)
    {
        printf("Iscritti alla prima %c: ", s);
        scanf(" ");
    }
    for(s='A'; s<='D'; s++)
    {
        printf("Iscritti alla seconda %c: ", s);
        scanf(" ");
    }
    for(s='A'; s<='D'; s++)
    {
        printf("Iscritti alla terza %c: ", s);
        scanf(" ");
    }
    for(s='A'; s<='D'; s++, media=0)
    {
        media = ;
        printf("\nLa media degli iscritti per classe
                nella sezione %c è: %f\n", s, media );
    }
}

```



## Esempio

### (stampa caratteri senza ripetizioni)

Si leggano da input alcuni caratteri alfabetici maiuscoli (hp: al massimo, 10) e si riscrivano in uscita evitando di ripetere caratteri già stampati.

**Soluzione:**

**while <ci sono caratteri da leggere>**

```

{
    ;
};

```

**while <ci sono elementi della struttura dati>**

**<stampa elemento>;**

→ Occorre una variabile dove memorizzare (senza ripetizioni) gli elementi letti in ingresso: il **vettore A**

```


```

→ il vettore A verrà riempito con al più 10 valori: definiamo la variabile **inseriti**, che rappresenterà la **"dimensione logica"** di A (cioè, il numero di elementi significativi):

```


```

→ **la dimensione fisica del vettore è 10, mentre la dimensione logica è data dal numero effettivo di caratteri diversi tra loro**

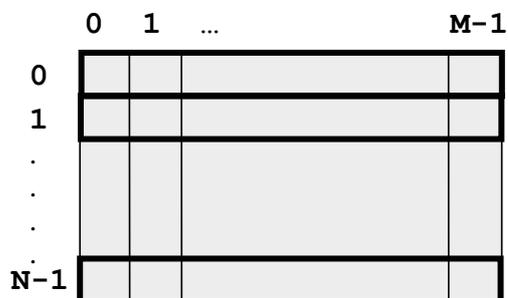
# Soluzione

```
#include <stdio.h>
main()
{ char A[10], c;
  int i, j, inseriti=0, trovato;
  printf( "\n Dammi 10 caratteri: " );
  for ( i=0; ( i< ); i++ )
  { scanf("%c", &c);
    /* verifica unicità: */
    trovato=0;
    for( j=0 ; ; j++ )
      if (c==A[j])
        trovato=1;
    /* se c non è ancora presente in A, lo inserisco: */
    if ( )
    {
    }
  }
  printf( "Inseriti %d caratteri \n", inseriti );
  for ( )
    printf( "%c\n", A[i] );
}
```

---

Fondamenti di Informatica L- A

## Matrici (vettori multidimensionali)



*matrice NxM*

---

Fondamenti di Informatica L- A

# Matrici (vettori multidimensionali)

Gli elementi di un vettore possono essere a loro volta di tipo vettore: in questo caso si parla di *matrici*

## Definizione di matrici:

```
<id-tipo> <id-variabile> [dim1] [dim2] .. [dimN];
```

## Significato:

- `<id-variabile>` è il nome di una variabile di tipo vettore di `dim1` componenti, ognuna delle quali è
  - un vettore di `dim2` componenti, ognuna delle quali è
    - un vettore di ..., ognuna delle quali è
      - un vettore di `dimN` componenti di tipo `<id-tipo>`.

→ Se le dimensioni sono  $N > 1$ , il vettore si dice **matrice N-dimensionale**

→ ('matrice' o per  $N=2$ ; 'matrice quadrata' se  $\text{dim1} = \text{dim2}$ )

Fondamenti di Informatica L- A

# Vettori multidimensionali

Ad esempio: *matrice* di float

```
float M[ 20 ][ 30 ];
```

→ M è un vettore di 20 elementi, ognuno dei quali è un vettore di 30 elementi, ognuno dei quali è un float.

**Accesso alle componenti:**  
il primo indice denota la *riga*, il secondo la *colonna*

```
M[0][0] = 7.1 ;  
M[1][29] = 0.5 ;  
M[19][29] = -1.99 ;
```

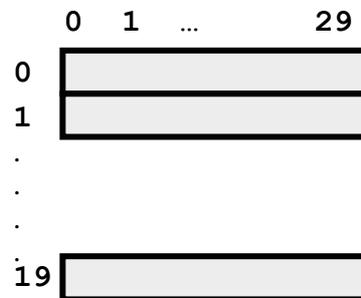
M[0]	0	1	...	29
0				
1				
·				
·			7.1	-1.99
·				
·			0.5	
·				
19				

M[19]                      M

→ **M[0]** rappresenta il primo vettore di 30 float (la prima *riga*)

Fondamenti di Informatica L- A

## Matrici “per righe”



*le colonne “non esistono”...*

---

Fondamenti di Informatica L- A

## Dichiarazione di tipo per vettori multi-dimensionali

```
typedef <id-tipo> <id-tipo-vettore>[dim1][dim2].. [dimN];
```

### Esempi:

```
typedef float MatReali [20] [30];  
MatReali Mat;  
/*Mat è un vettore di venti elementi, ognuno dei quali  
è un vettore di trenta reali; quindi: Mat è una  
matrice di 20 X 30 di reali*/
```

In alternativa si puo` fare:

```
typedef float VetReali[30];  
typedef VetReali MatReali[20];  
MatReali Mat;
```

---

Fondamenti di Informatica L- A

# Inizializzazione di matrici

Anche nel caso di vettori multi-dimensionali l'inizializzazione si può effettuare in fase di definizione, tenendo conto che, in questo caso, gli elementi sono a loro volta vettori:

## Esempio:

```
int matrix[4][4] = { {1,0,0,0},
                    {0,1,0,0},
                    {0,0,1,0},
                    {0,0,0,1} };
```

→ La memorizzazione avviene "per righe":

	0	1	2	3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

Si può anche omettere il numero di righe:

```
int matrix[][4] = {{1,0,0,0}, {0,1,0,0}, {0,0,1,0}, {0,0,0,1}};
```

---

Fondamenti di Informatica L- A

## Esempio

(lettura e stampa di matrici)

```
#include <stdio.h>
#define R 10
#define C 25

typedef float matrice[R][C];
main()
{
    matrice M;
    int i, j;
    /* lettura: */
    for(i=0; i<R; i++)
        for(j=0; j<C; j++)
            { printf("M[%d][%d]? ", i, j);
              scanf("%f", &M[i][j]); }
    /* stampa: */
    for(i=0; i<R; i++)
        { for(j=0; j<C; j++)
          printf("%f\t", M[i][j]);
          printf("\n"); }
}
```

---

Fondamenti di Informatica L- A

# Esempio

## (prodotto di due matrici)

Si realizzi un programma che esegua il prodotto (righe x colonne) di 2 matrici a valori reali.

**Definizione:**

date due matrici rettangolari  $A[N][L]$  e  $B[L][M]$ , il risultato di  $AxB$  e' una matrice  $C[N][M]$  tale che:

$\forall i \in [0, N-1], \forall j \in [0, M-1]:$

$$C[i,j] = \sum_{(k=1..L)} A[i][k]*B[k][j]$$

Rappresentazione dei dati:

```
/*costanti per le dimensioni delle matrici:*/
#define N 2
#define M 3
#define L 4

float A[N][L];
float B[L][M];
float C[N][M];
```

---

Fondamenti di Informatica L- A

## Soluzione

```
#include <stdio.h>
#define N 2
#define M 3
#define L 4
main()
{ float A[N][L];
  float B[L][M];
  float C[N][M];
  int i, j, k;
  /* inizializzazione di A e B */
  printf("Elementi di A?\n");
  for (i=0; i<N; i++)
  { printf("\nRiga %d (%d elementi): ", i, L);
    for (j=0; j<L; j++)
      scanf( );
  }

  printf("Elementi di B?\n");
  for (i=0; i<L; i++)
  { printf("\nRiga %d (%d elementi): ", i, M);
    for (j=0; j<M; j++)
      scanf( );
  } /* continua... */
```

---

Fondamenti di Informatica L- A

```
/* ...prodotto matriciale: */
```

```
for (i=0; i<N; i++)  
  for (j=0; j<M; j++)  
  {  
    [REDACTED]  
  }
```

```
/* stampa del risultato: */
```

```
for (i=0; i<N; i++)  
  {  
    printf("\nC[%d]:\t", i);  
    [REDACTED]  
  }
```

---

Fondamenti di Informatica L- A

## Esempio

### ordinamento di un vettore

Dati n valori interi forniti in ordine casuale, stampare in uscita l'elenco dei valori dati in ordine crescente.

→ E' necessario mantenere in memoria tutti i valori dati per poter effettuare i confronti necessari: utilizziamo i vettori.

#### Soluzione:

Esistono vari procedimenti risolutivi per questo problema (algoritmi di ordinamento, o **sorting**):

- **naïve-sort**
- bubble sort
- quick sort
- merge sort
- ...

**Risolveremo il problema utilizzando il Metodo dei Massimi successivi (*naïve-sort*).**

---

Fondamenti di Informatica L- A

# Ordinamento di un vettore mediante *naïve sort*

## Descrizione dell' algoritmo:

Dato un vettore: `int V[dim];`

1. eleggi un elemento come massimo temporaneo (`V[max]`)
2. confronta il valore di `V[max]` con tutti gli altri elementi del vettore (`V[i]`):  
`se V[i] > V[max] , max=i`
3. quando hai finito i confronti, scambia `V[max]` con `V[dim-1]`; il massimo ottenuto dalla scansione va in ultima posizione.
4. riduci il vettore di un elemento (`dim=dim-1`) e, se `dim > 1`, torna a 1.

Fondamenti di Informatica L- A

## naïve sort con vettore di dimensione `dim = 5`

	V	V	V	V	V
V[0]	10	10	10	2	2
V[1]	3	3	3	3	3
V[2]	15	15	2	10	10
V[3]	38	2	15	15	15
V[4]	2	38	38	38	38
	<b>v[0]..v[3]</b>	<b>v[0]..v[2]</b>	<b>v[0]..v[1]</b>	<b>v[0]</b>	

Fondamenti di Informatica L- A

```

#include <stdio.h>
#define dim 10
main()
{ int V[dim], i,j, max, tmp, quanti;
  /* lettura dei dati */
  for ( i=0; i<dim; i++ )
  {   printf("valore n. %d: ",i);
      scanf("%d", &V[i]);
  }
  /*ordinamento */
  for( ciclo di n= dim-1 iterazioni )
  {   determinazione del
      valore massimo presente
      nella porzione di vettore

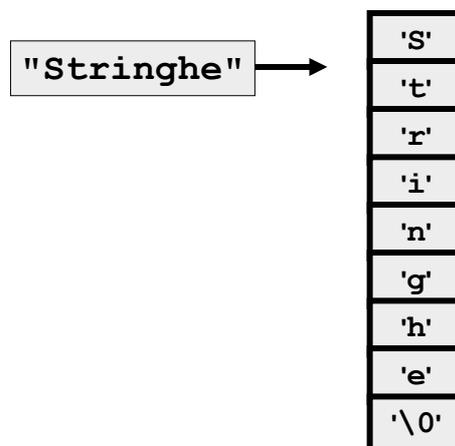
      se ho trovato un valore maggiore
      di quello presente nell'ultima cella
      effettuo uno scambio
  }
  /* stampa del vettore ordinato per esercizio */

```

---

Fondamenti di Informatica L- A

## Stringhe (vettori di caratteri)




---

Fondamenti di Informatica L- A

# Vettori di caratteri: le stringhe

Una stringa è un vettore di caratteri, manipolato e gestito secondo la **convenzione**:

**l'ultimo elemento significativo di ogni stringa è seguito dal carattere nullo '\0'**  
**'\0' corrisponde al codice ASCII zero**

È responsabilità del programmatore gestire tale struttura in modo consistente con il concetto di stringa (ad esempio, garantendo la presenza del terminatore '\0').

**Ad esempio:**

```
char A[10]={'b','o','l','o','g','n','a','\0'};
```

'b'	'o'	'l'	'o'	'g'	'n'	'a'	'\0'	?	?
-----	-----	-----	-----	-----	-----	-----	------	---	---

oppure:

```
char A[10]="bologna"; /* il terminatore '\0' è aggiunto automaticamente */
```

**Letture di stringhe (formato %s):**

```
char B[25];
```

```
scanf("%s", &B); /* la sequenza di caratteri letta viene assegnata a B;
                  il terminatore '\0' è aggiunto automaticamente */
```

---

Fondamenti di Informatica L- A

## gets & puts

La scanf (con formato %s) prevede come separatore anche il blank (spazio bianco):

→ è possibile soltanto leggere stringhe che non contengono bianchi;

**Ad esempio, se l'input è:**

*Nel mezzo del cammin di nostra vita,  
mi ritrovai per una selva oscura...*

...

**Leggendo con:**

```
char s1[80], s2[80];
scanf("%s", &s1); /* s1 vale "Nel" */
scanf("%s", &s2); /* s2 vale "mezzo" */
```

→ la scanf non è adatta a leggere intere linee (che possono contenere spazi bianchi, caratteri di tabulazione, etc.).

Per questo motivo, in C esistono funzioni specifiche per fare I/O di linee:

- gets (legge fino a '\n')
- puts (scrive e aggiunge '\n')

---

Fondamenti di Informatica L- A

## gets

è una funzione standard, che legge una intera riga da input, fino al primo carattere di fine linea ('`\n`', newline) e l'assegna a una stringa.

```
gets(char str[]);
```

assegna alla stringa `str` i caratteri letti.  
Il carattere '`\n`' viene sostituito (nella stringa di destinazione `str`) da '`\0`'.

**Ad esempio, dato l'input:**

*Nel mezzo del cammin di nostra vita,  
mi ritrovai per una selva oscura...*

**Leggendo con:**

```
char S[80]; gets(S);
```

S vale:

```
"Nel mezzo del cammin di nostra vita,"
```

---

Fondamenti di Informatica L- A

## puts

è una funzione standard che scrive una stringa sull'output aggiungendo un carattere di fine linea ('`\n`', newline):

```
puts(char str[]);
```

**Ad esempio:**

```
char S1[80]="Dante Alighieri";  
char S2[80]="La Divina Commedia";  
puts(S1);  
puts(S2);
```

stampa sullo standard output:

```
Dante Alighieri  
La Divina Commedia
```

→ In generale: `puts(S)`; è equivalente a `printf("%s\n", S)`;

---

Fondamenti di Informatica L- A

## Esempio: lunghezza di una stringa

Programma che calcola la **lunghezza** di una stringa (cioè, il numero di caratteri significativi).

```
/* lunghezza di una stringa */
#include <stdio.h>
main()
{ char str[81]; /* str ha al max. 80 caratteri*/
  int i;
  printf("\nImmettere una stringa:");
  scanf("%s",&str); /* %s formato
                    stringa */
  for (i=0; str[i]!='\0'; i++);
  printf("\nLunghezza: \t %d\n",i);
}
```

→ La `scanf` legge i caratteri in ingresso fino al primo separatore (bianco, newline, ecc.) e li assegna al vettore `str` aggiungendo il terminatore di stringa.

**NB:** La lunghezza di una stringa si può anche calcolare utilizzando la funzione standard di libreria `strlen()` (previa inclusione di `string.h`)

---

Fondamenti di Informatica L- A

## Esempio: concatenamento di stringhe

Programma che concatena due stringhe `s1` e `s2` date: il risultato viene inserito in `s1`.

```
#include <stdio.h>
/* concatenamento di due stringhe */
main()
{
  char s1[81], s2[81];
  int j,i;
  printf("\nPrima stringa: \t");
  scanf("%s",&s1);
  printf("\nSeconda stringa: \t");
  scanf("%s",&s2);
  for ( j=0 ; ; j++ );
  printf("\nStringa: \t%s\n",s1);
}
```

**NB:** il concatenamento di due stringhe si può anche ottenere utilizzando la funzione standard di libreria `strcat()` (previa inclusione di `string.h`)

---

Fondamenti di Informatica L- A

## string.h

Il C fornisce una libreria standard di funzioni per la gestione di stringhe; per poterla utilizzare è necessario includere il file header `string.h`:

```
#include <string.h>
```

### Lunghezza di una stringa:

```
int strlen(char str[ ]);
```

→ restituisce la lunghezza (cioè, il numero di caratteri significativi) della stringa `str` specificata come argomento.

### Ad esempio:

```
char S[10]= "bologna";  
int k;  
k=strlen(S);      /* k vale 8 */
```

---

Fondamenti di Informatica L- A

## string.h

### Concatenamento di 2 stringhe:

```
strcat( char str1[], char str2[] );
```

→ concatena le 2 stringhe date `str1` e `str2`. Il risultato del concatenamento è in `str1`.

### Ad esempio:

```
char S1[20] = "reggio";  
char S2[20] = "emilia";  
strcat( S1, S2 ); /* S1 = "reggioemilia" */
```

### Copia di stringhe:

```
strcpy(char str1[], char str2[]);
```

→ copia la stringa `str2` in `str1`.

### Ad esempio:

```
char S1[10] = "Riccardo";  
char S2[10];  
strcpy( S2, S1 ); /* S2 = "Riccardo" */
```

---

Fondamenti di Informatica L- A

## string.h

### Confronto di 2 stringhe:

```
int strcmp(char str1[ ], char str2[ ]);
```

→ esegue il confronto tra le due stringhe date str1 e str2. Restituisce:

- 0 se le due stringhe sono identiche;
- un valore negativo (ad esempio, -1), se str1 precede str2 (in ordine lessicografico);
- un valore positivo (ad esempio, +1), se str2 precede str1 (in ordine lessicografico).

#### Ad esempio:

```
char S1[10] = "bologna";  
char S2[10] = "napoli";  
int k;  
k = strcmp(S2,S1); /* k = 0 */  
k = strcmp(S1,S2); /* k = -1 */  
k = strcmp(S2,S2); /* k = 0 */
```

---

Fondamenti di Informatica L- A

## string.h

**strlen(char s[])**

→ |s|

**strcpy(char s1[], char s2[])**

→ (assegnamento) s1=s2

**strcat(char s1[], char s2[])**

→ (somma) s1 = s1+s2

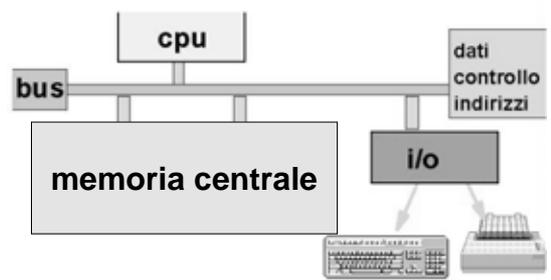
**strcmp(char s1[], char s2[])**

→ (s1 == s2) ? 0 : ( (s1 < s2) ? -1 : +1 );

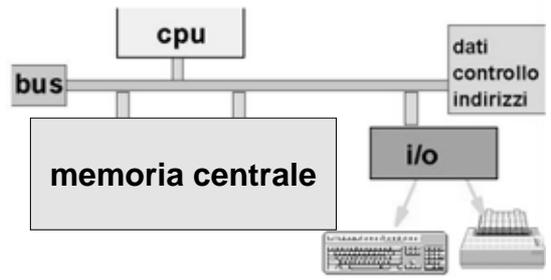
---

Fondamenti di Informatica L- A

# Architettura di un elaboratore



# ARCHITETTURA DI UN ELABORATORE



Ispirata al modello della **Macchina di Von Neumann** (Princeton, Institute for Advanced Study, anni '40).



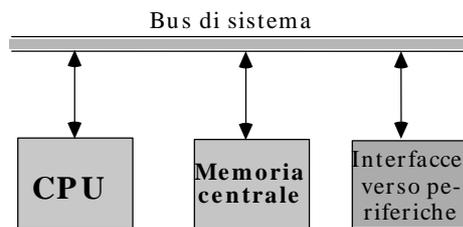
**John von Neumann** (Neumann János)  
(28/12/1903 – 8/2/1957)



**EDVAC** (Electronic Discrete Variable Automatic Computer)

*First Draft of a Report on the EDVAC. 30/06/1945*

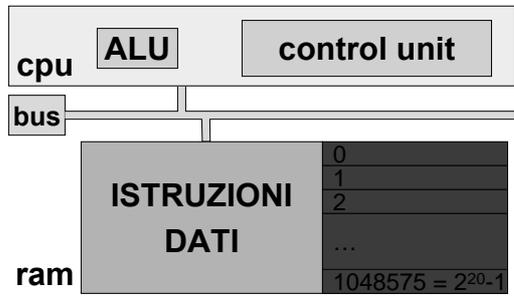
## ARCHITETTURA DI VON NEUMANN



### UNITÀ FUNZIONALI fondamentali

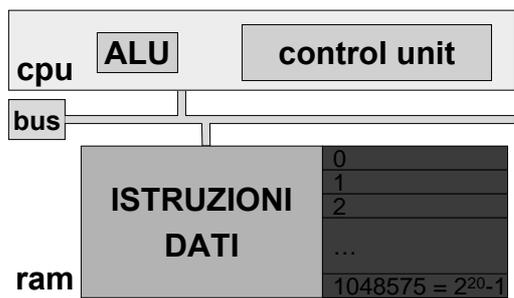
- **Processore (CPU)**
- **Memoria Centrale (RAM & ROM)**
- **Periferiche (ingresso / uscita)**
- **Bus di sistema**

# CPU & MEMORIA



- ALU (Arithmetic & Logic Unit)
- Unità di Controllo
- Registri

# UNITÀ DI ELABORAZIONE (CPU)



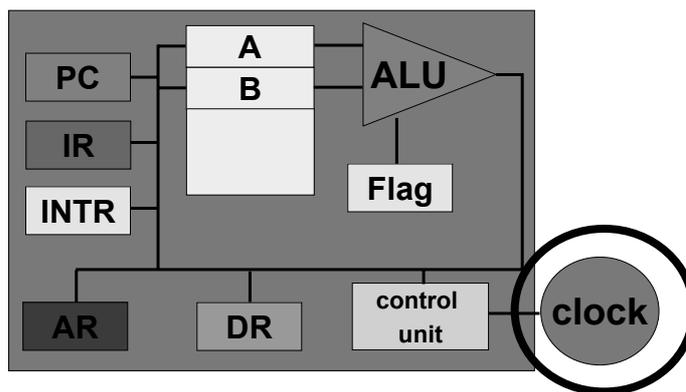
**ALU (Arithmetic / Logic Unit)**  
 Eseguce le operazioni aritmetiche e logiche elementari

## UNITÀ DI ELABORAZIONE (CPU)



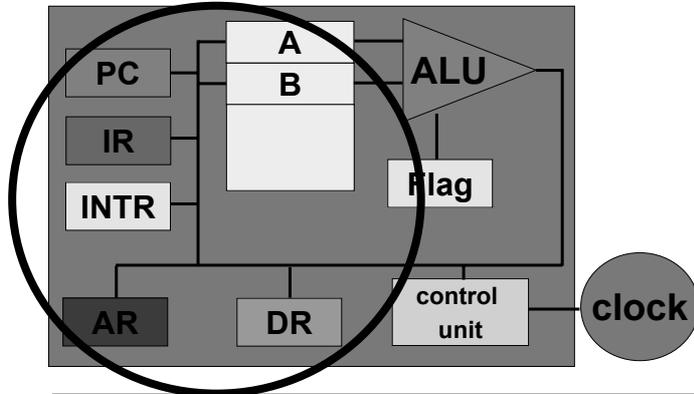
**Unità di Controllo (Control Unit):** controlla e coordina l'attività della CPU. (In particolare, controlla il trasferimento dei dati tra memoria e registri e la decodifica e l'esecuzione delle istruzioni)

## UNITÀ DI ELABORAZIONE (CPU)



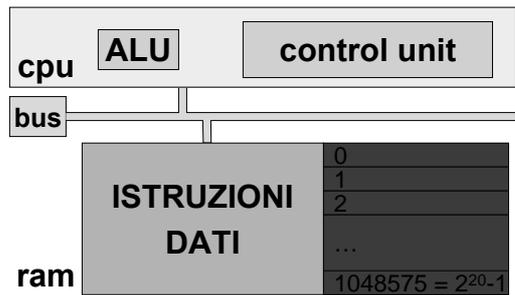
**Il clock** dà la base dei tempi necessaria per mantenere il sincronismo fra le operazioni

## UNITÀ DI ELABORAZIONE (CPU)



I registri (qui A, B, PC, Flag,...) sono *locazioni* usate per memorizzare dati, istruzioni, o indirizzi **all'interno della CPU**. L'accesso ai registri è *molto veloce*.

## UNITÀ DI ELABORAZIONE (CPU)



La **memoria centrale** è una **collezione di celle numerate**, che possono contenere **DATI e ISTRUZIONI**. Le istruzioni sono disposte in memoria in **celle di indirizzo crescente**.

## UNITÀ DI ELABORAZIONE (CPU)

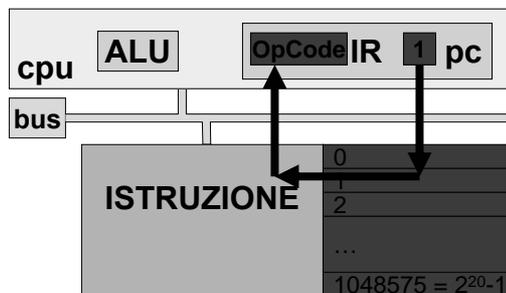


L'unità di controllo fa funzionare l'elaboratore.  
 Da quando viene acceso a quando è spento, essa esegue in continuazione il ciclo di *prelievo / decodifica / esecuzione* (fetch / decode / execute) .

## IL CICLO fetch / decode / execute

### FETCH

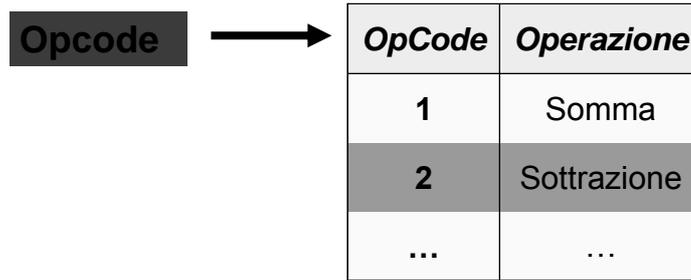
- si accede alla **prossima istruzione** (cella il cui indirizzo è contenuto nel registro **PC**) ...
- ... e **la si porta dalla memoria centrale**, memorizzandola nel **Registro Istruzioni (IR)**



# IL CICLO fetch / decode / execute

## DECODE

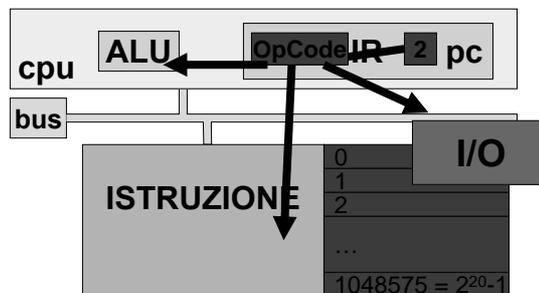
- si decodifica il tipo dell'istruzione in base al suo *OpCode* (codice operativo)



# IL CICLO fetch / decode / execute

## EXECUTE

- si individuano i dati usati dall'istruzione
- si trasferiscono tali dati nei registri opportuni
- si esegue l'istruzione.



# IL CICLO fetch / decode / execute

## ATTENZIONE

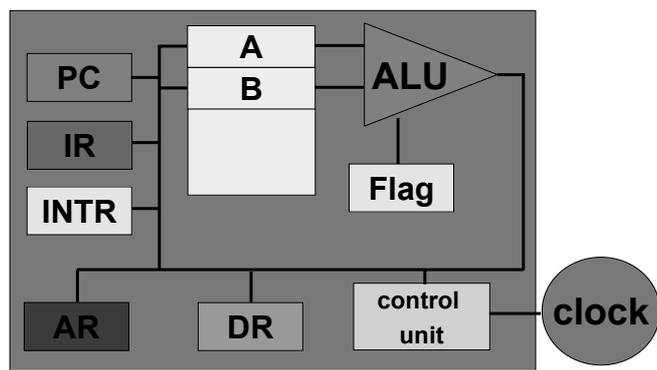
Istruzioni particolari possono *alterare il prelievo delle istruzioni da celle consecutive*:

- istruzioni di **salto**
- istruzioni di **chiamata a sotto-programmi**
- istruzioni di **interruzione**

---

Fondamenti di Informatica L- A

# REGISTRI

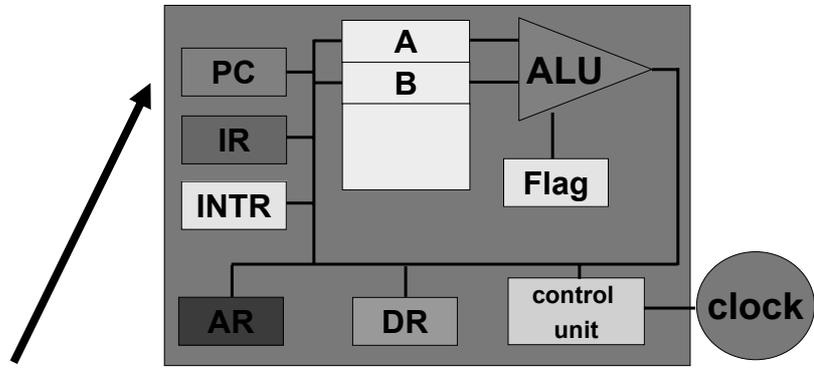


**I registri sono *locazioni di memoria interne a CPU*, e come tali *molto veloci*.**

---

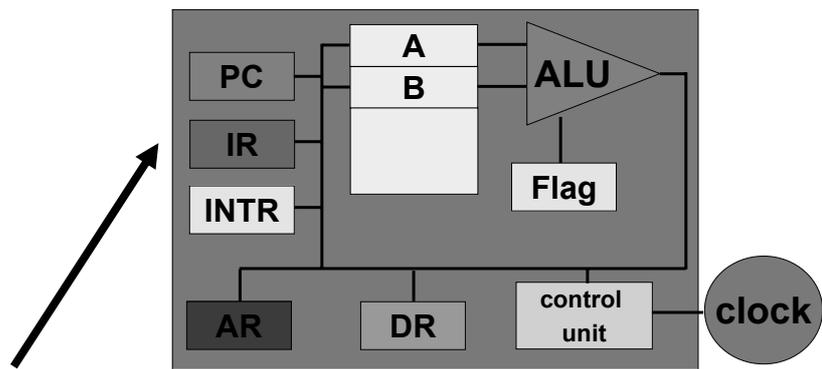
Fondamenti di Informatica L- A

# REGISTRI



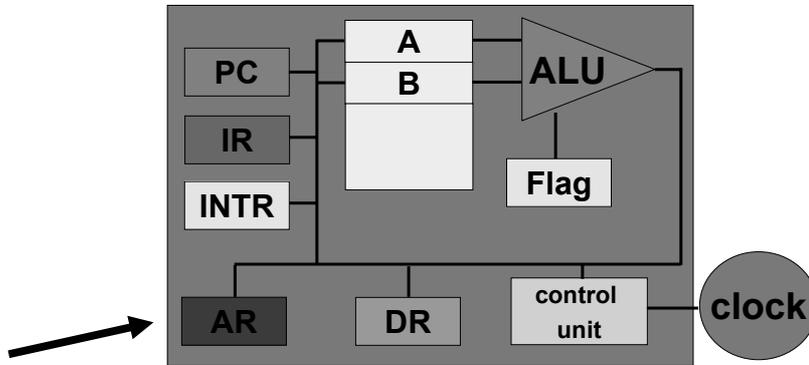
**Program Counter (PC)**  
 Indica l'indirizzo della cella di memoria che contiene la prossima istruzione da eseguire

# REGISTRI



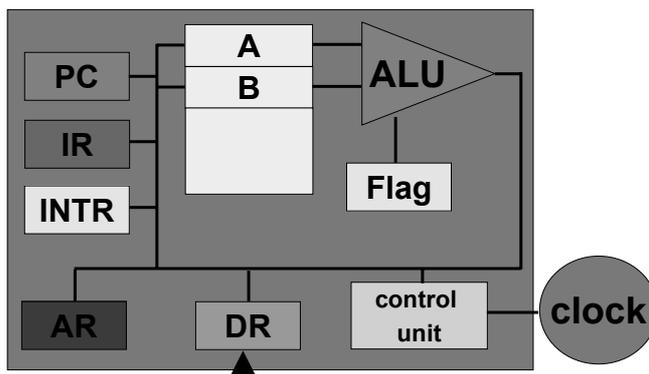
**Instruction Register (IR)**  
 Contiene l'istruzione da eseguire.

# REGISTRI



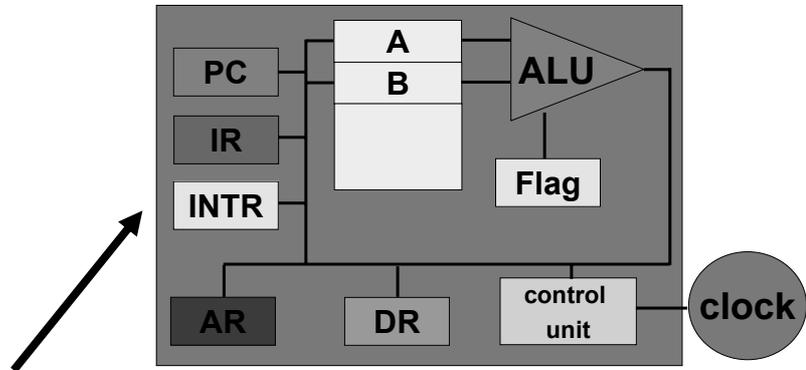
**Registro Indirizzi (Address Register, AR)**  
 Contiene l'indirizzo della cella di memoria da selezionare per il trasferimento di un dato con la CPU

# REGISTRI



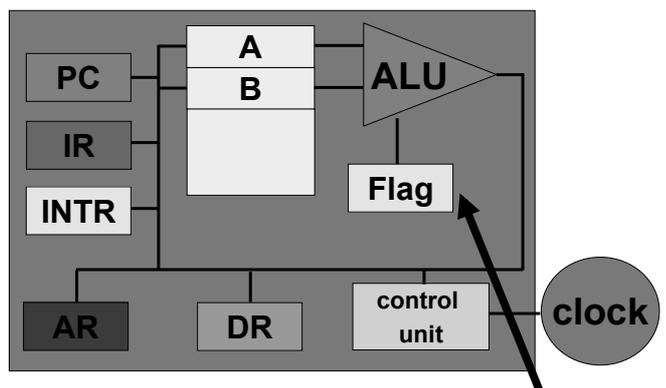
**Registro Dati (Data Register, DR)**  
 Contiene il dato attualmente oggetto di elaborazione

# REGISTRI



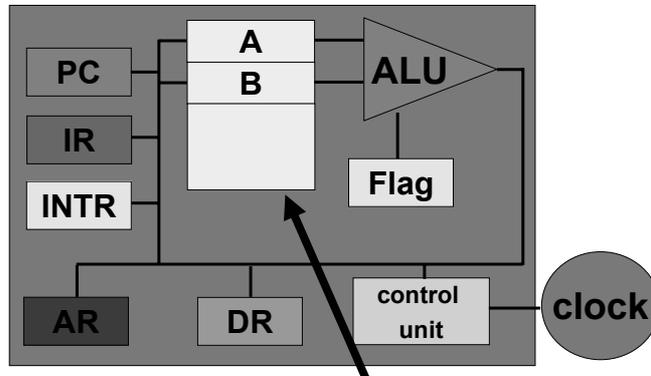
**Registro Interruzioni (INTerrupt Register)**  
 Serve per scopi particolari (non discussi)

# REGISTRI



**Registro dei Flag (Flag)**  
 Ogni flag indica la presenza/assenza di una proprietà nell'ultimo risultato generato dalla ALU. Altri bit riassumono lo stato del processore.

# REGISTRI



**Registri di uso generale (A,B,C,...)**  
 Sono usati per contenere dati (in particolare, operandi/risultati di operazioni aritmetico/logiche)

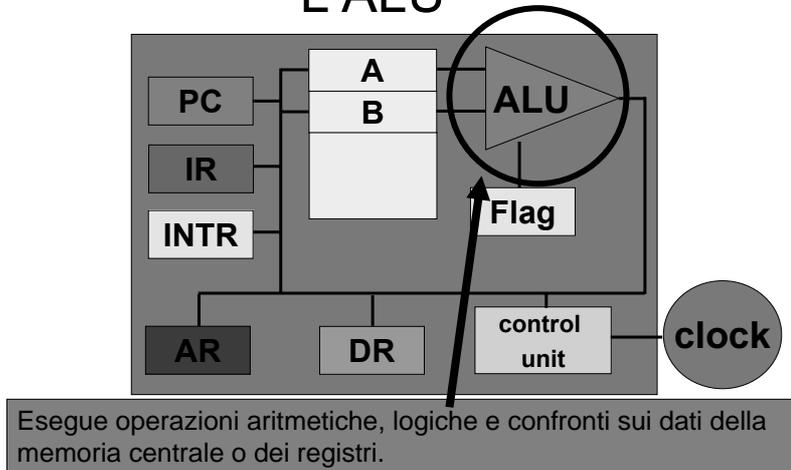
# MULTITASKING

Poiché i registri compendiano *tutto lo stato dell'elaborazione* di un certo processo,

- salvando in memoria il contenuto di tutti i registri è possibile **accantonare un processo per passare a svolgerne un altro**
- ripristinando dalla memoria il contenuto di tutti i registri precedentemente salvati è possibile **ripristinare lo stato di un processo accantonato, riprendendone l'esecuzione come se nulla fosse accaduto ("context switch")**.

Questa possibilità è ciò che consente a un sistema operativo di eseguire *più compiti* "allo stesso tempo"

## L'ALU



## L'ALU (segue)

### ESEMPIO SEMPLICE:

ALU in grado di eseguire **somma**, **sottrazione**, **prodotto**, **divisione** con due operandi contenuti nei registri A e B.

1. I due operandi vengono caricati nei registri A e B;
2. La ALU viene attivata da un comando inviato dalla CPU che specifica il tipo di operazione;
3. Nel registro A viene caricato il risultato dell'operazione eseguita dalla ALU;
4. Il registro FLAG riporta sui suoi bit indicazioni sul risultato dell'operazione (riporto, segno, etc.).



Alterazione di due bit nel registro **Flag**:  
**carry** (riporto) e **sign** (segno)

# LA MEMORIA CENTRALE

## INDIRIZZAMENTO

- E' l'attività con cui l'elaboratore seleziona una particolare cella di memoria
- Per farlo, l'elaboratore pone l'indirizzo della cella desiderata nel Registro Indirizzi (AR).
  - **se AR è lungo  $N$  bit, si possono indirizzare  $2^N$  celle di memoria** (numerate da 0 a  $2^N-1$ )
  - esempio:  $N=10 \Rightarrow 1024$  celle.
- **Oggi, AR è lungo tipicamente 32 / 40 bit**  
 → **SPAZIO INDIRIZZABILE di 4 GB / 1 TB**

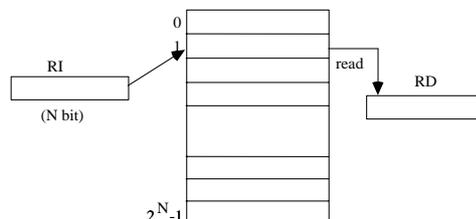
---

Fondamenti di Informatica L- A

# LA MEMORIA CENTRALE (2)

## OPERAZIONI

- **Lettura (*Read*)**: il contenuto della cella di memoria indirizzata dal Registro Indirizzi è copiato nel Registro Dati.



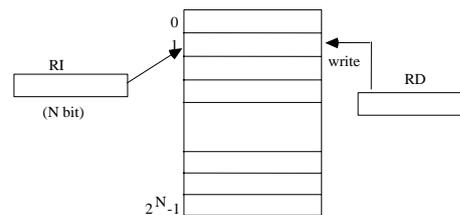

---

Fondamenti di Informatica L- A

## LA MEMORIA CENTRALE (3)

### OPERAZIONI

- **Scrittura (*Write*)**: il contenuto del Registro Dati è copiato nella cella di memoria indirizzata dal Registro Indirizzi.



## DISPOSITIVI DI MEMORIA

### DISPOSITIVI FISICI

- **RAM**: Random Access Memory (ad accesso casuale): su di essa si possono svolgere operazioni sia di lettura che di scrittura
- **ROM**: Read Only Memory (a sola lettura): non volatili e non scrivibili dall'utente (che la ordina con un certo contenuto); in esse sono contenuti i dati e programmi per inizializzare il sistema
- **PROM**: Programmable ROM. Si possono scrivere soltanto una volta, mediante particolari apparecchi (detti programmatori di PROM).

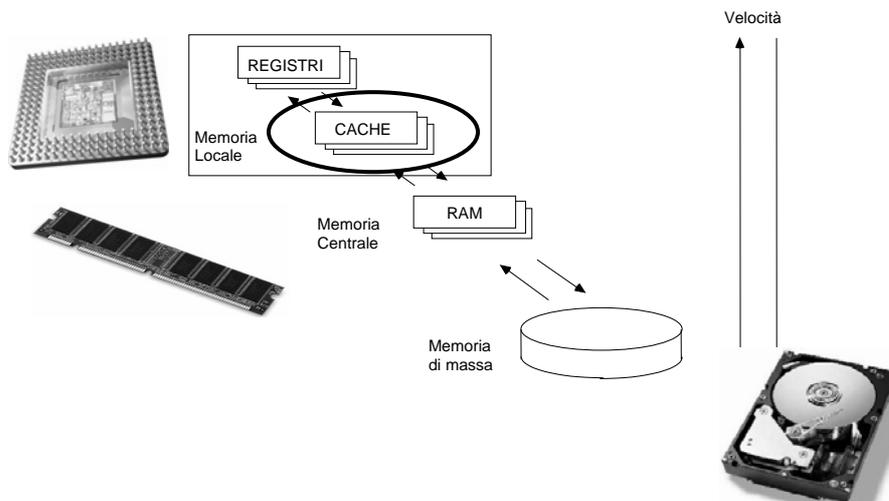
# DISPOSITIVI DI MEMORIA

## DISPOSITIVI FISICI

- **EPROM**: Erasable-Programmable ROM (si cancellano sottoponendole a raggi ultravioletti).
- **EEPROM**: Electrically-Erasable-PROM (si cancellano elettricamente).

Il **Firmware** è costituito da software memorizzato su ROM, EPROM, etc. (codice microprogrammato).

# GERARCHIA DELLE MEMORIE



## MEMORIA CACHE

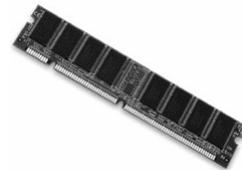
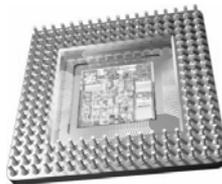
### PROBLEMA:

Grande differenza tra la “larghezza di banda” di CPU (2GHz) e RAM (10<sup>2</sup> MHz)

Vale a dire: sebbene la RAM sia veloce, ha comunque prestazioni molto inferiori rispetto a quelle della CPU

### CONSEGUENZA:

il processore *perde tempo* ad aspettare l'arrivo dei dati dalla RAM.




---

Fondamenti di Informatica L- A

## MEMORIA CACHE

### SOLUZIONE:

Inserire **tra processore e RAM** una **memoria particolarmente veloce** dove tenere i dati usati più spesso (**memoria cache**)

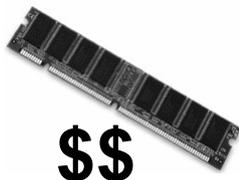
### In questo modo,

- ◆ **la prima volta** che il microprocessore carica dei dati dalla memoria centrale, tali dati vengono caricati anche sulla cache
- ◆ **le volte successive**, i dati possono essere letti dalla cache (veloce) invece che dalla memoria centrale (più lenta)

---

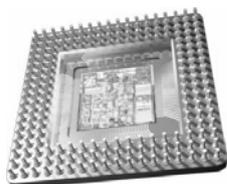
Fondamenti di Informatica L- A

# MEMORIA CACHE



**DUBBIO:**

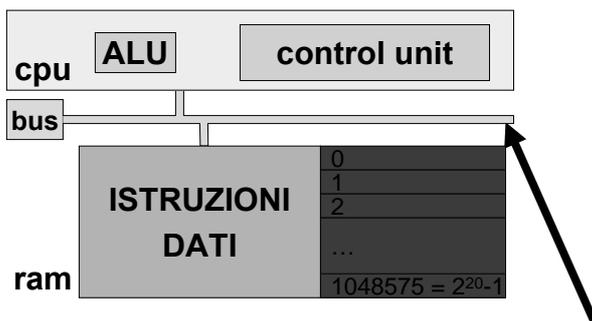
Ma se abbiamo memorie così veloci, *perché non le usiamo per costruire tutta la RAM?*



Semplice...  
perché costano molto!!

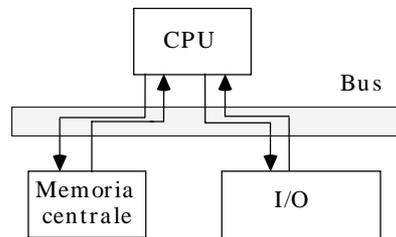
**OGGI**, la cache è spesso già presente dentro al processore (**cache di 1° e 2° livello**), e altra può essere aggiunta (**cache di 3° livello**)

# BUS DI SISTEMA



**Il Bus di Sistema** interconnette la CPU, le memorie e le interfacce verso dispositivi periferici (I/O, memoria di massa, etc.)

## BUS DI SISTEMA

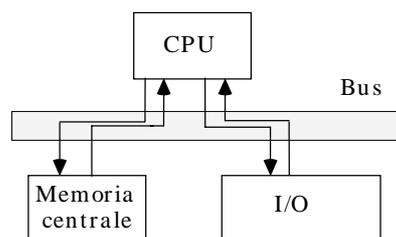


Il Bus collega **due unità funzionali alla volta**:

- una trasmette...
- ... e l'altra riceve.

Il trasferimento dei dati avviene o *sotto il controllo della CPU*, o mediante *accesso diretto alla memoria (DMA)*.

## BUS DI SISTEMA



Il Bus è in realtà **un insieme di linee diverse**:

- bus dati (*data bus*)
- bus indirizzi (*address bus*)
- bus comandi (*command bus*)

## BUS DI SISTEMA

### BUS DATI

- **bidirezionale**
- serve per trasmettere dati **dalla memoria o viceversa**.

### BUS INDIRIZZI

- **unidirezionale**
- serve per trasmettere **il contenuto del registro indirizzi alla memoria**  
(si seleziona una specifica cella su cui viene eseguita o un'operazione di lettura o una operazione di scrittura)

---

Fondamenti di Informatica L- A

## BUS DI SISTEMA

### BUS COMANDI

- **bidirezionale**
- tipicamente usato per **inviare comandi verso la memoria** (es: lettura o scrittura) o **verso una periferica** (es. stampa verso la stampante → interfaccia)
- può essere usato per **inviare comandi verso il processore** nel caso di DMA (o interfacce di I/O)

---

Fondamenti di Informatica L- A

## INTERFACCE DI I/O




---

Fondamenti di Informatica L- A

## OLTRE la macchina di Von Neumann

- **Problema:** nella Macchina di Von Neumann le operazioni sono **strettamente sequenziali**.
- Altre soluzioni introducono forme di **parallelismo**
  - **processori dedicati** (*coprocessori*) al calcolo numerico, alla gestione della grafica, all'I/O.
  - **esecuzione in parallelo** delle varie fasi di un'istruzione: mentre se ne esegue una, si acquisiscono e decodificano le istruzioni successive (**pipeline**)
  - **architetture completamente diverse:** sistemi multi-processore, macchine dataflow, LISP machine, reti neurali.

---

Fondamenti di Informatica L- A