Corso di Informatica A.A. 2009-2010

Lezione 2

La macchina di Von Neumann

Lo schema della macchina di Von Neumann, modello sul quale si basano praticamente tutti i moderni computer è basato su quattro elementi fondamentali:

- La memoria
- L'unità aritmetico logica (ALU)
- L'unità di controllo
- Le unità di Ingresso/Uscita (I/O)

Processore

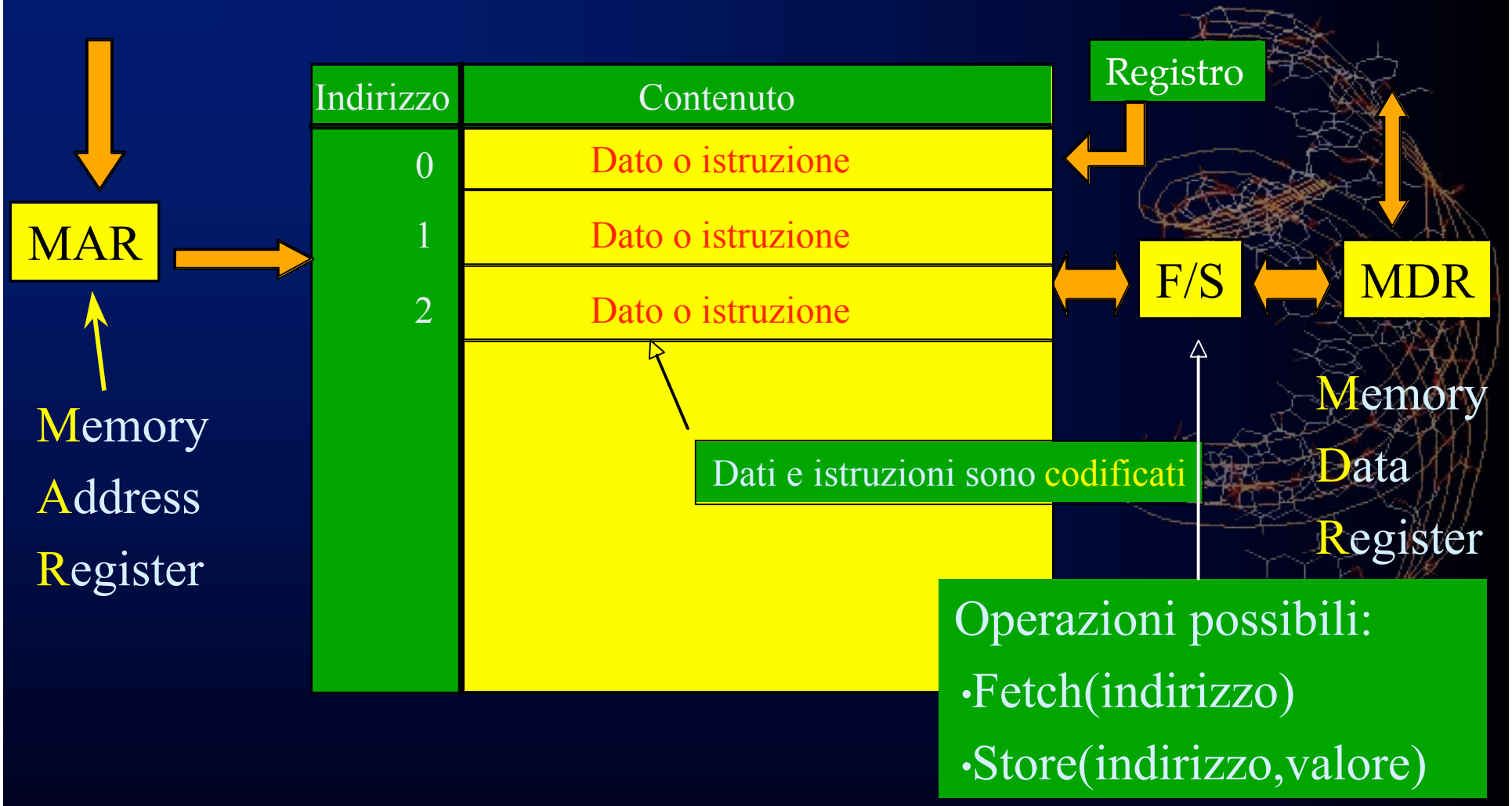
Memoria e cache

La memoria è l'unità funzionale che memorizza e recupera le informazioni che devono essere elaborate

La memoria di un calcolatore utilizza il cosiddetto **accesso random** ⟹ **RAM (Random Access Memory)**

- memoria divisa in celle di dimensioni fisse cui è associato un indirizzo
- gli accessi avvengono utilizzando gli indirizzi
- il tempo per l'accesso è lo stesso per tutte le celle (qualche decina di nanosecondi)

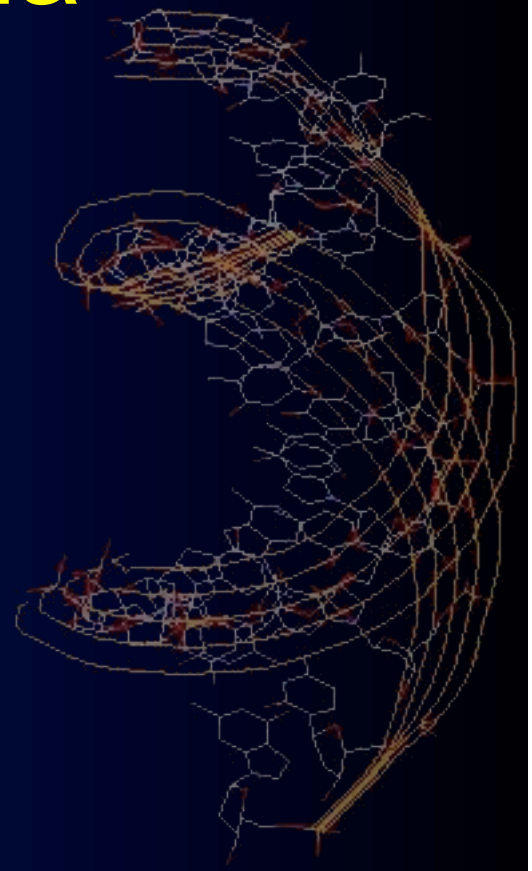
La memoria RAM



I registri di memoria

Fetch (indirizzo):

1. Indirizzo ➔ registro MAR
2. Decodifica indirizzo
3. Copia contenuto ➔ registro MDR



Store (indirizzo, valore):

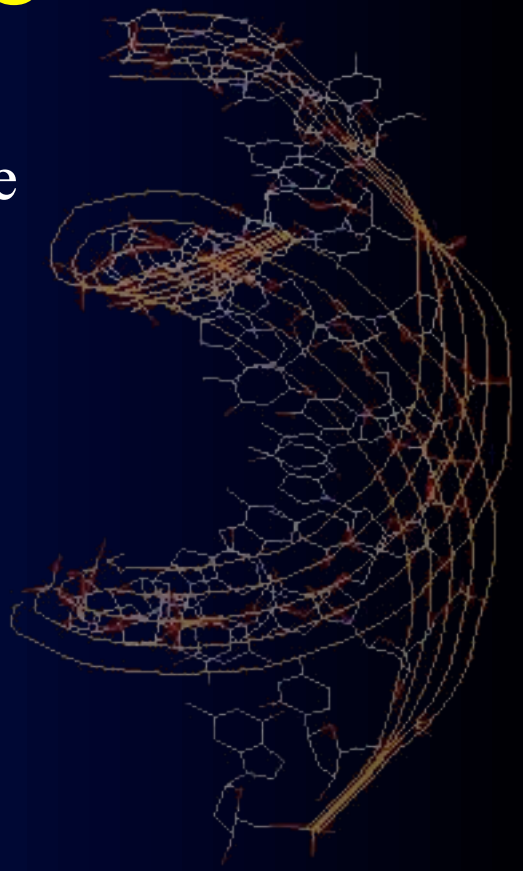
1. Indirizzo ➔ registro MAR
2. Valore ➔ registro MDR
3. Decodifica indirizzo
4. Scrive contenuto registro MDR ➔ locazione memoria

La memoria cache

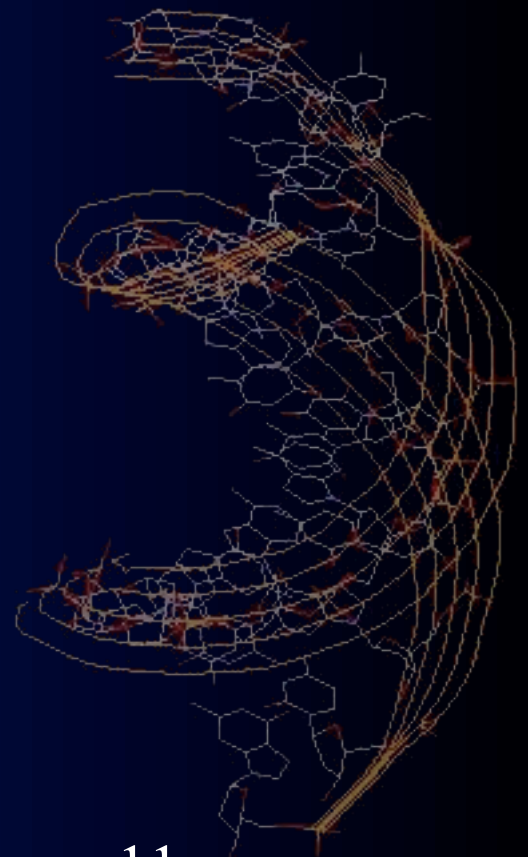
Principio di località: quando un programma esegue il fetch di un dato vi è un'alta probabilità che nell'immediato futuro acceda allo stesso dato

Memoria cache

5-10 volte più veloce della RAM,
ma molto più "piccola"



Esempio



- tempo medio accesso alla RAM = 20 nsec
- tempo medio di accesso alla cache = 5 nsec
- tasso di successo della cache = 70 %

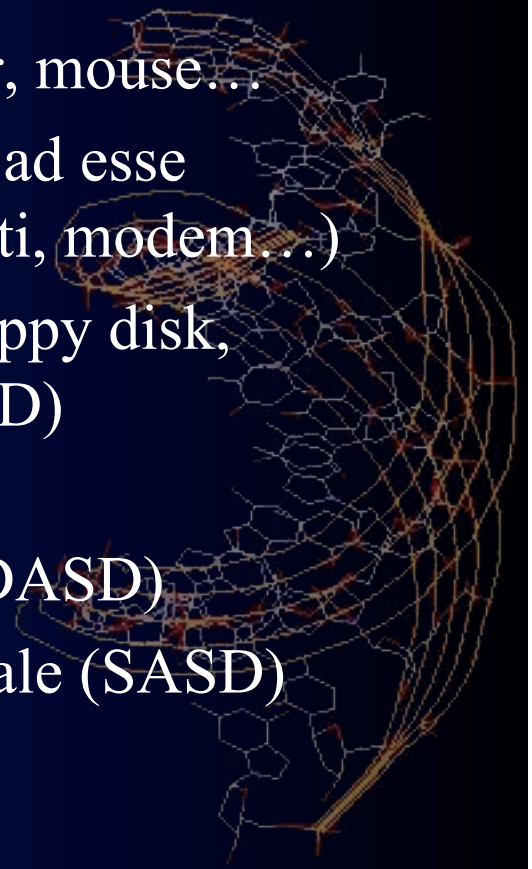
Tempo medio di accesso = (0.7 * 5) + 0.3 *(5+20) nsec = 11 nsec

Le unità di I/O

- Dispositivi di interfaccia utente: tastiera, monitor, mouse…
- Dispositivi di comunicazione: porte e dispositivi ad esse connessi (stampanti, modem…)
- Dispositivi di memoria di massa: (Hard disk, Floppy disk, CDROM, DVD)

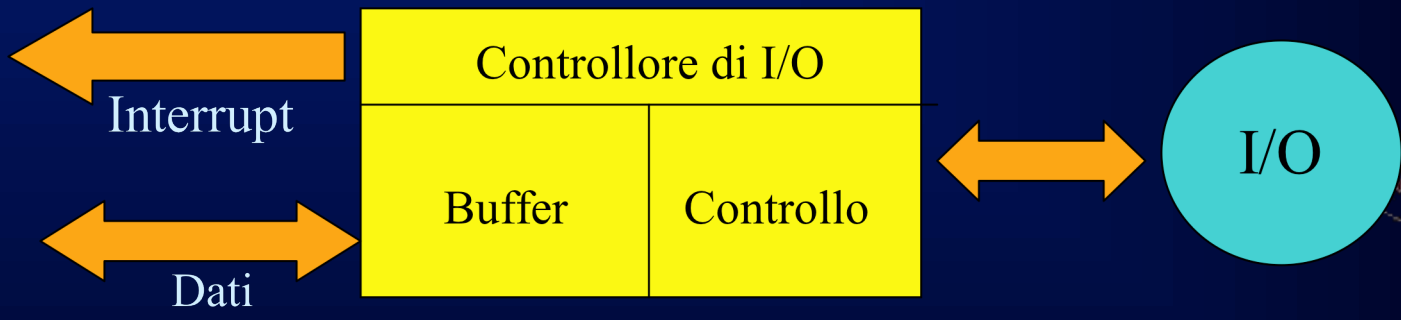
Dispositivi di memorizzazione ad accesso diretto (DASD)

Dispositivi di memorizzazione ad accesso sequenziale (SASD)



Le unità di I/O

Tempi caratteristici di accesso: diversi ordini di grandezza più lenti della memoria RAM

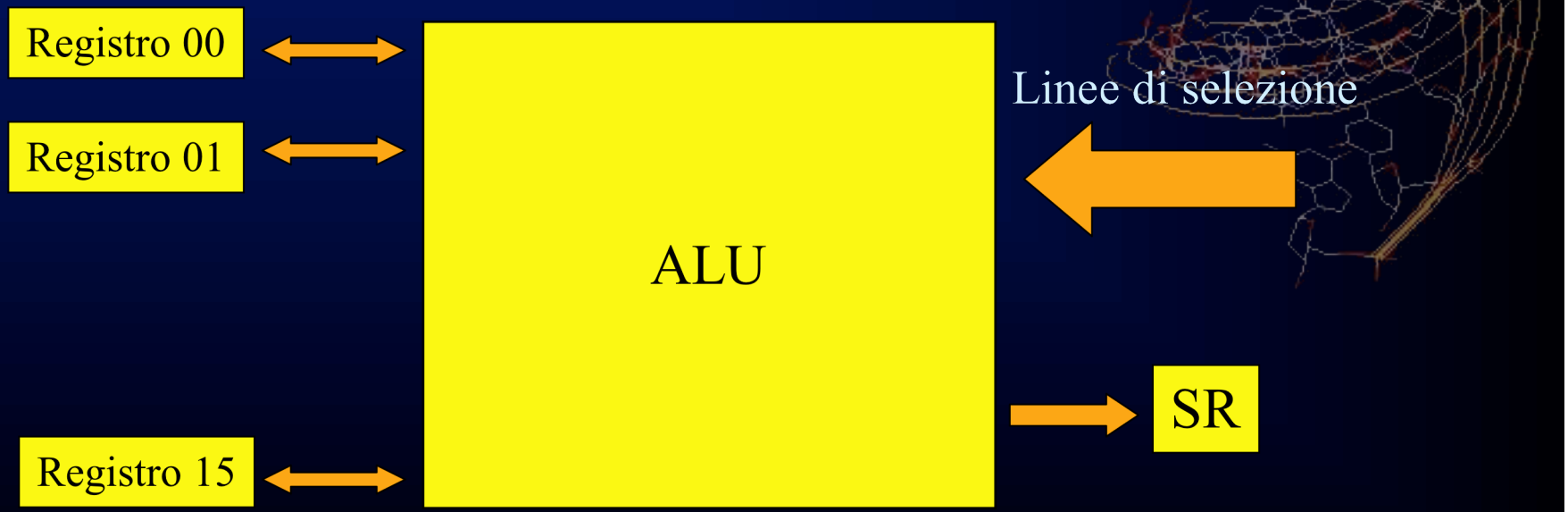


L'unità aritmetico logica: ALU

Il cuore calcolante del computer: effettua un insieme finito e predeterminato di operazioni matematiche e logiche.

Gli operandi vengono letti da registri in ingresso, e il risultato dell'operazione è scritto su un registro in uscita.

Il registro di stato (SR) riporta il segno del risultato e la presenza di riporto o di una condizione di errore.

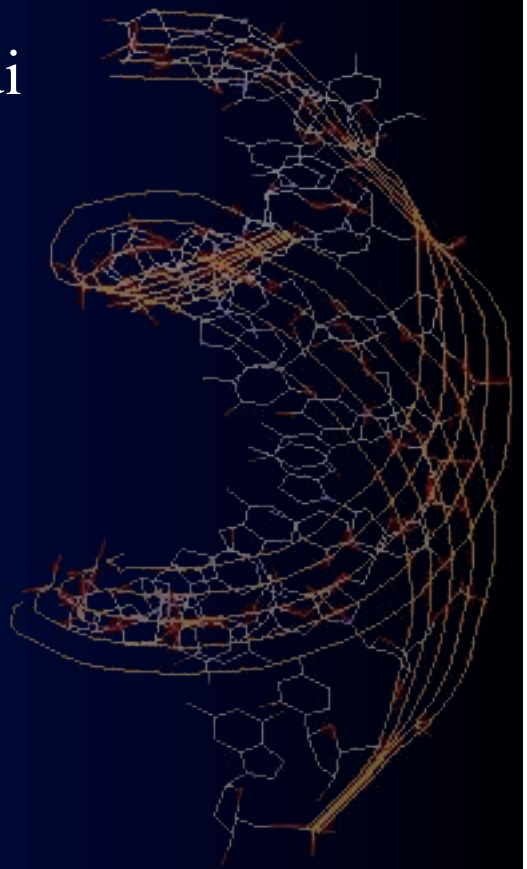


L'unità di controllo

Caratteristica fondamentale dell'architettura di Von Neumann: programma memorizzato

Ciclo di esecuzione di un programma:

- Instruction fetch
- Decodifica
- Esegui



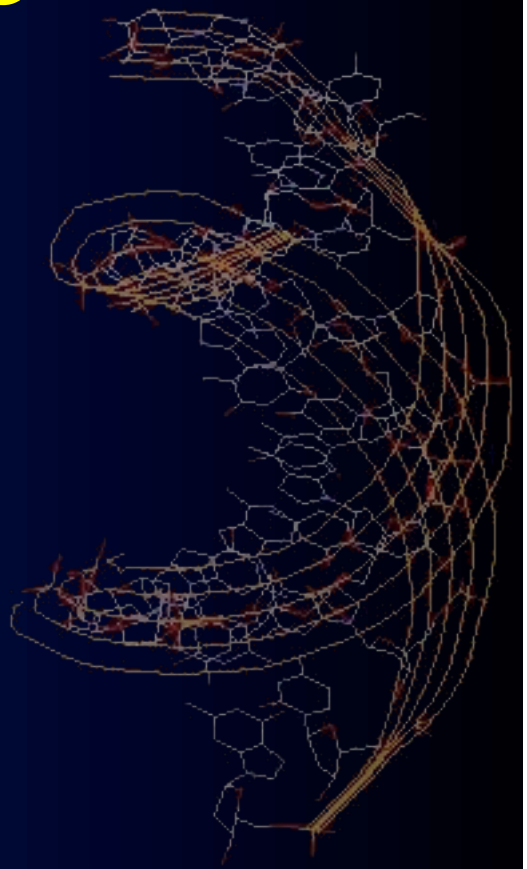
Struttura di una istruzione in linguaggio macchina:

Codice operativo – Indirizzo 1 – Indirizzo 2

Esempio:

ADD X,Y (Y=X+Y)

L’unità di controllo



L’insieme di tutte le operazioni che possono Essere eseguite da un processore si chiama **set di istruzioni**

Differenti processori posseggono differenti set di istruzioni

L’unità di controllo

Riassumendo

La struttura di Von Neumann permette di calcolare una sequenza di istruzioni opportunamente codificate e memorizzate e di controllare il flusso dell'esecuzione. La gestione dell'I/O è delegata a opportuni controllori per ottimizzare le prestazioni.

La condizione chiave per la realizzazione del calcolatore è disporre di un sistema efficiente e affidabile di codifica dell'informazione, ovvero dei dati e delle istruzioni che devono essere via via eseguite.

Oltre Von Neumann ?

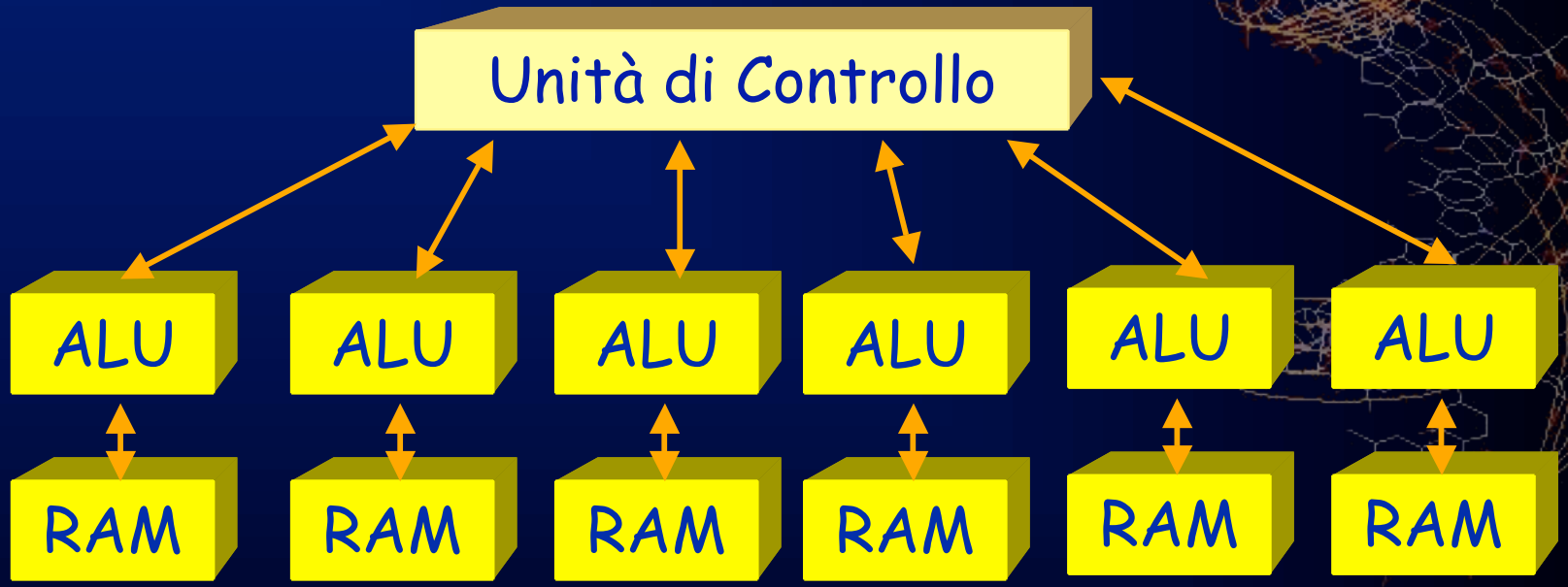
Lo schema di Von Neumann domina la scena dei calcolatori da ormai più di mezzo secolo…un tempo lunghissimo in un campo come l'informatica.

L'unico "punto debole" della macchina di Von Neumann è la sua intrinseca sequenzialità.

I tentativi per superare l'approccio alla Von Neumann si basano tutti su una qualche forma di parallelismo.

Architettura SIMD

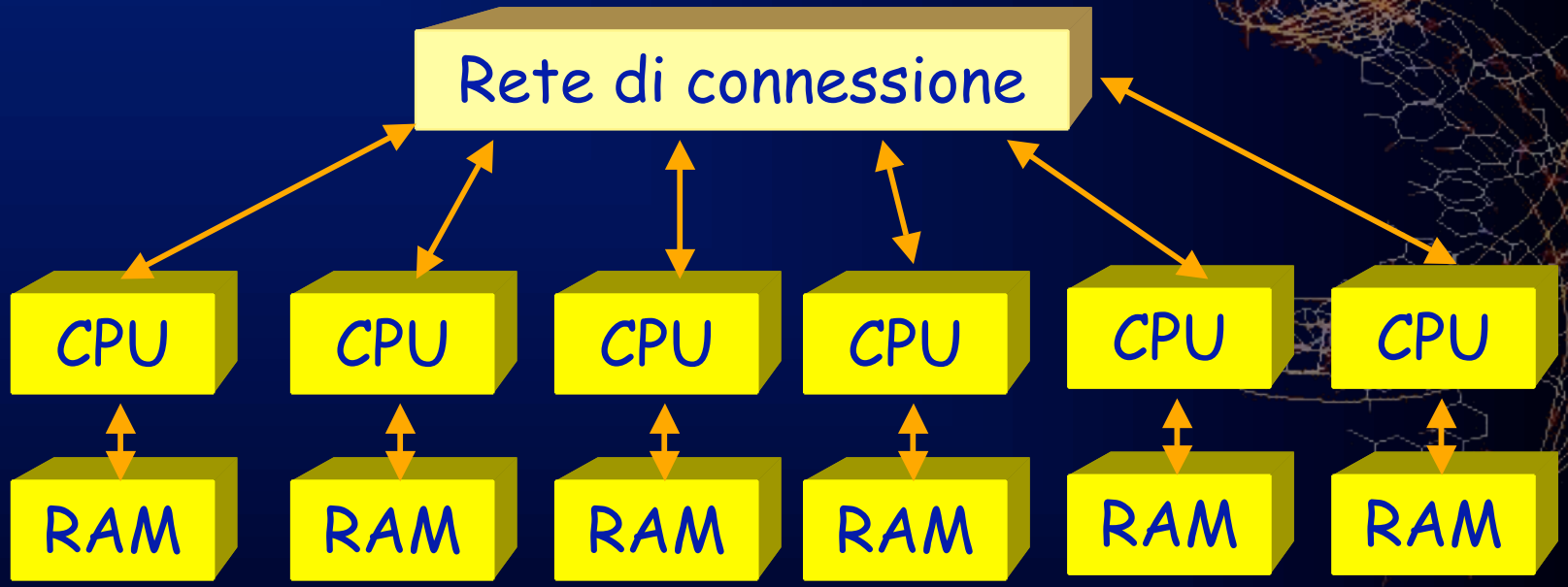
Il primo tipo di architettura parallela ad essere implementato è stato lo schema Single Instruction Multiple Data stream



Molto efficace per operazioni su dati di tipo vettoriale

Architettura MIMD

Uno schema di parallelismo più complesso, e potenzialmente più potente è quello detto di Multiple Instruction Multiple Data stream



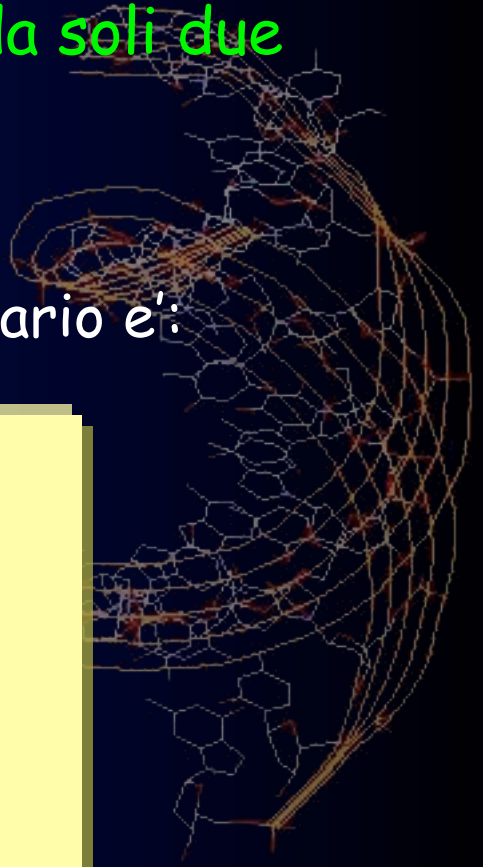
Per essere realmente utile richiede la realizzazione di *algoritmi paralleli*

Codifica binaria dell'informazione

- Rappresentazioni posizionali
- Codifica numeri interi
- Codifica numeri in virgola mobile

La codifica binaria

L'alfabeto piu' semplice e' quello costituito da soli due simboli.

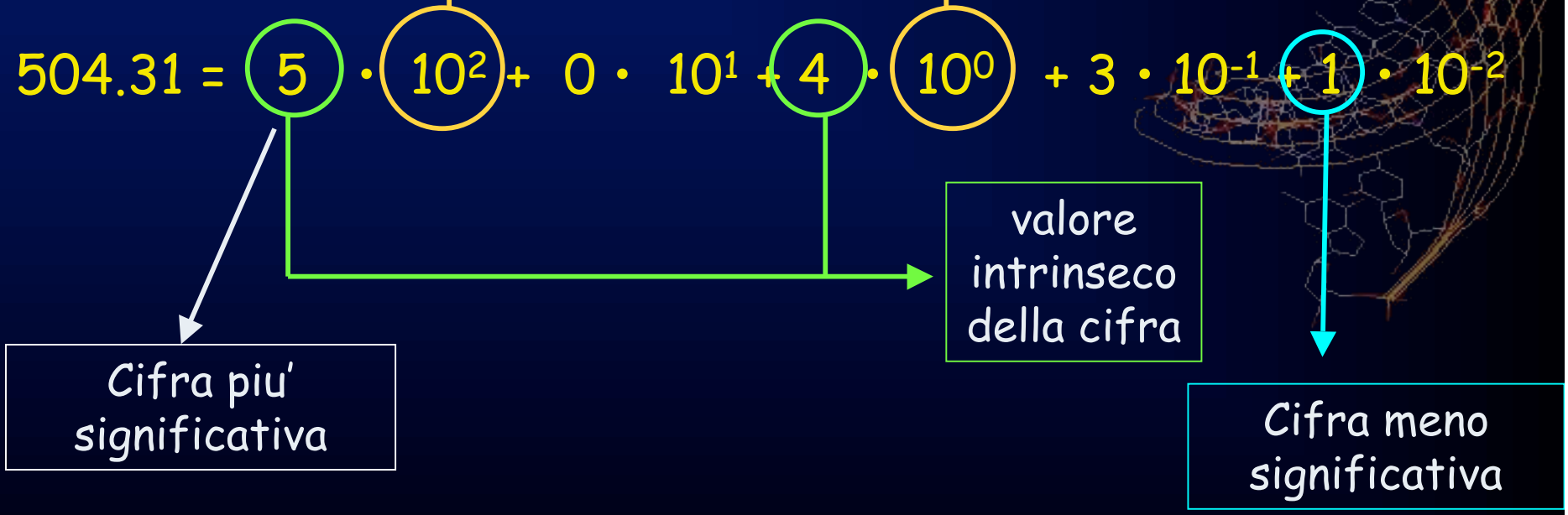


Un sistema automatico basato su un alfabeto binario e':

- ✓ Facilmente implementabile su un supporto fisico
  - ⇨ Sostanze magnetiche con due opposte polarizzazioni
  - ⇨ Passaggio o meno di corrente
  - ⇨ Passaggio o meno di luce
  - ⇨ . . . .
- ✓ Piu' economico
- ✓ Piu' affidabile

Sistema di numerazione decimale

1. E' posizionale (unita', decine, centinaia)
2. E' costituito da 10 cifre (base = 10)



Notazione posizionale

La notazione posizionale consente di scrivere un numero *N* di una certa base generica *b* come una sequenza di cifre.

$$504.31 = 5 \cdot 10^2 + 0 \cdot 10^1 + 4 \cdot 10^0 + 3 \cdot 10^{-1} + 1 \cdot 10^{-2}$$

Generalizzando, data una base *b* ed un insieme di cifre:

$$a_{n-1}\, a_{n-2}\, a_{n-3} \ldots a_2\, a_1\, a_0 \cdot a_{-1}\, a_{-2} \ldots\ldots a_{-m-2}\, a_{m-2}\, a_{m-1}\, a_{-m}$$

Valore decimale del numero N

$$V(N) = \sum_{i=-m}^{n-1} a_i b^i$$

$$0 \le a_i \le b-1$$

Sistemi posizionali

Sistema	Base	Cifre
binario	2	0 1
ottale	8	0 1 2 3 4 5 6 7
decimale	10	0 1 2 3 4 5 6 7 8 9
esadecimale	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

$(208)_{10}=(11010000)_2=(D0)_{16}=(320)_8$

base

Limiti della rappresentazione posizionale (I)

Data un sistema di numerazione di base b qual e' il massimo numero intero positivo rappresentabile in una sequenza di n cifre?

Es. in 4 cifre il sistema decimale puo' rappresentare interi positivi da 0 a 9999, ovvero 10^4 valori diversi, ovvero b^4 valori.

Formalmente, il numero massimo N_{max} rappresentabile con n cifre nella base b e':

$$N_{\max} = \sum_{i=0}^{n-1} (b-1)\, b^i = b^n - 1$$

Limiti della rappresentazione posizionale (II)

Quante cifre sono invece necessarie per rappresentare un dato numero in una base?
Basta invertire la relazione precedente:

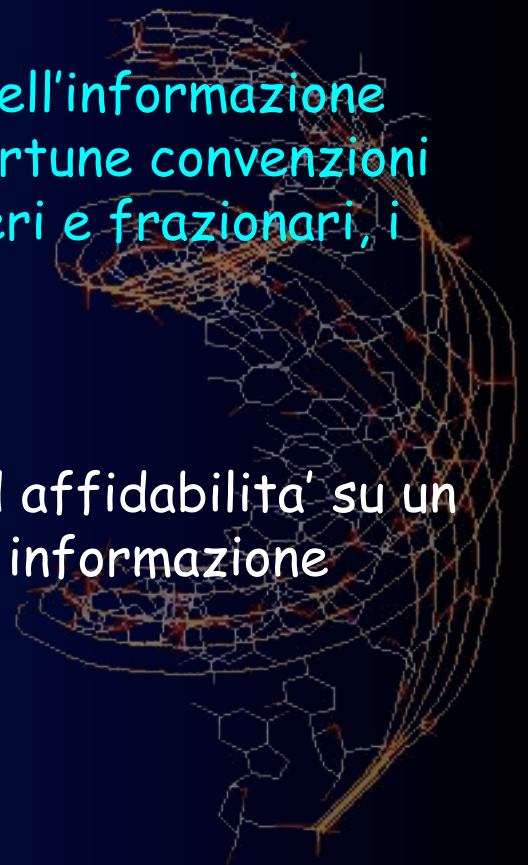
Ceiling: minimo intero superiore

$$n_{\min} = \lceil \log_b (N+1) \rceil$$

Es. in una sequenza di 16 cifre il sistema binario puo' rappresentare 2^{16} = 65536 interi positivi che vanno da 0 a 65535;
il numero minimo di cifre per rappresentare il numero 1000 è ceil($\log_2$(1001)) = 10

Il computer e l'informazione

Il sistema posizionale binario permette la codifica dell'informazione numerica attraverso l'uso di solo due cifre. Con opportune convenzioni si possono rappresentare anche i numeri relativi interi e frazionari, i caratteri, le immagini ed i suoni.



A causa della maggiore facilita' implementativa ed affidabilita' su un supporto fisico, i calcolatori digitali trattano solo informazione codificata in forma binaria.

L'unita' elementare d'informazione manipolata e memorizzata da un computer e' detta *bit* (binary digit), e può assumere i valori 1 e 0.

Alcune convenzioni

byte → sequenza di 8 bit (Es. 01001011)

word → da 16 a 64 bit a seconda dell'architettura; convenzionalmente 16bit

longword → convenzionalmente 32 bit

Multiplo	Sigla	Valore	Approssimazione
Kilo	k	$2^{10} = 1024$	$\approx 10^3$
Mega	M	$2^{20} = 1024^2$	$\approx 10^6$
Giga	G	$2^{30} = 1024^3$	$\approx 10^9$
Tera	T	$2^{40} = 1024^4$	$\approx 10^{12}$
Peta	P	$2^{50} = 1024^5$	$\approx 10^{15}$
Exa	E	$2^{60} = 1024^6$	$\approx 10^{18}$

Conversione decimale-binario

18 : 2 = 9 resto 0

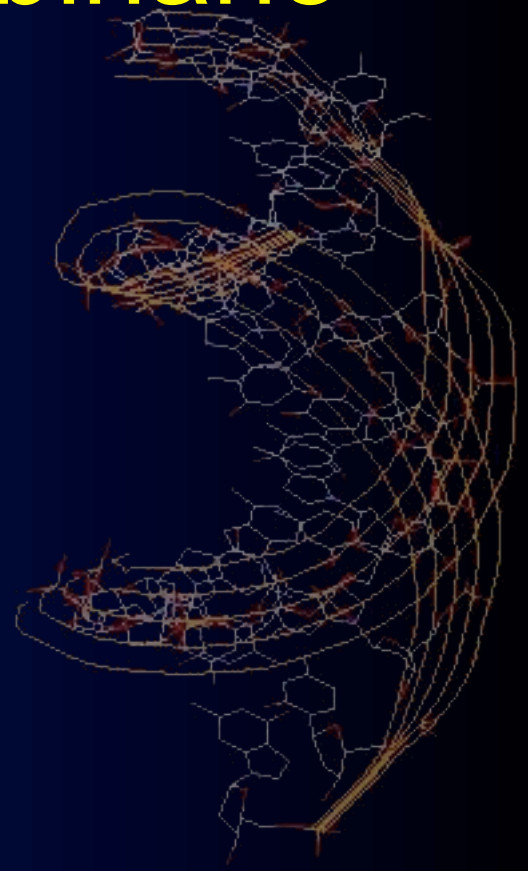
9 : 2 = 4 resto 1

4 : 2 = 2 resto 0

2 : 2 = 1 resto 0

1 : 2 = 0 resto 1

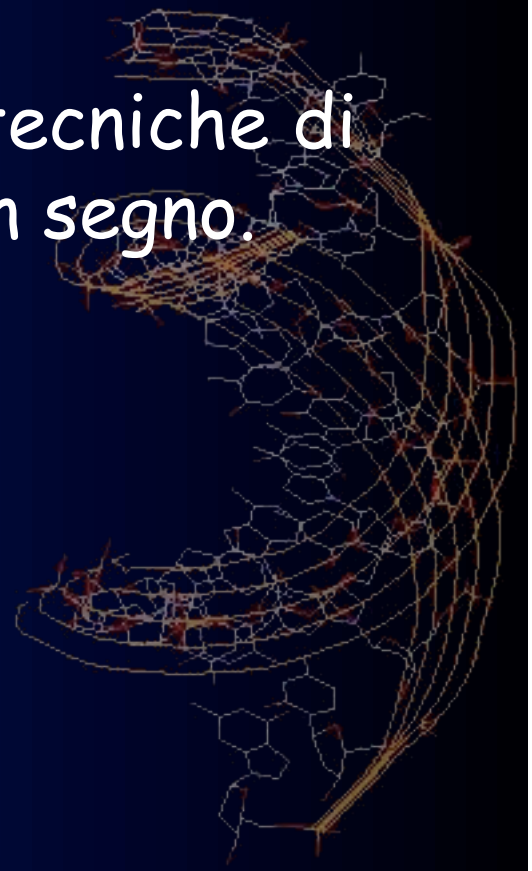
1 0 0 1 0



Codifica dei numeri interi

Nell' aritmetica binaria vi sono più tecniche di organizzazione dei numeri interi con segno.

1. Modulo e segno;
2. Complemento alla base;
3. .....
4. .....



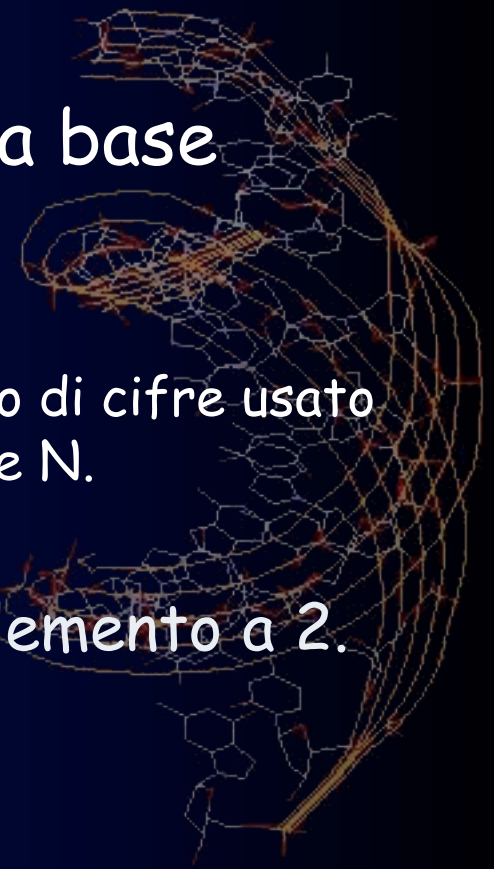
Complemento alla base

Dato un numero N il complemento alla base $C_b(N)$ e' dato da:

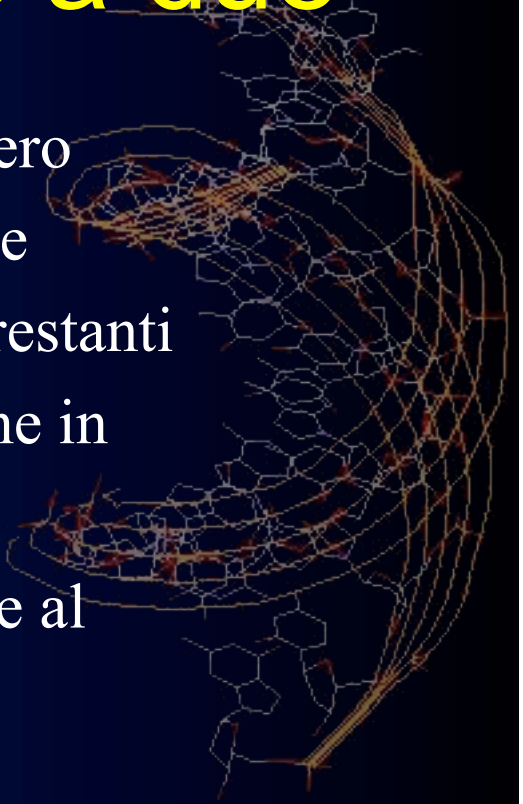
$$C_b(N) = b^n - N$$

dove n e' il numero di cifre usato per rappresentare N.

Per b = 2, il complemento e' noto come complemento a 2.



Rappresentazione dei numeri negativi in complemento a due



Il bit più significativo rappresenta il segno del numero

La rappresentazione di un numero positivo si ottiene codificando il valore assoluto del numero con i bit restanti

La rappresentazione di un numero negativo si ottiene in tre passi

1. Si rappresenta il numero positivo corrispondente al valore assoluto del numero negativo da codificare
2. Si invertono tutti i bit (0→1, 1→0)
3. Si somma 1 al risultato ottenuto al passo 2

Esempio

Utilizziamo una codifica a 4 bit.

La codifica di +5 è 0101

La codifica di -5 si ottiene invece in tre passi:

1. Si parte dalla rappresentazione del valore assoluto

0101

2. Si invertono tutti i bit

1010

3. Si somma 1 ottenendo la codifica desiderata

1011

Numeri in virgola mobile (I)

La rappresentazione degli interi non esaurisce certo l'insieme di numeri rappresentabili. Supponiamo inoltre di voler rappresentare un intero molto grande, come ad esempio duemila miliardi: N = 2 000 000 000 000

Avremmo bisogno di ben 41 bit per rappresentarlo come unsigned (o 42 come int).

La maggior parte dello spazio sarebbe sprecato per conservare informazione sugli zeri: in effetti si può usare la *notazione scientifica* e scrivere, in modo molto più compatto:

$$N = 2.0 \cdot 10^{12}$$

Numeri in virgola mobile (II)

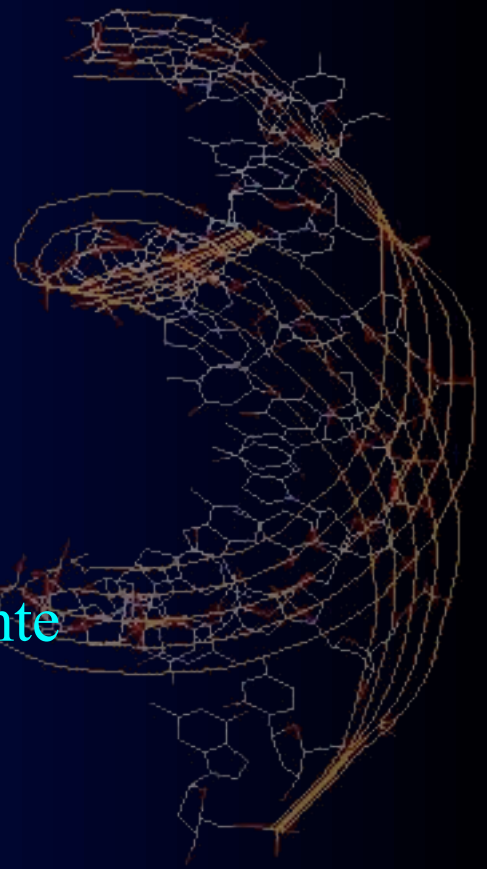
Notazione:

$$N = 2.0 \cdot 10^{12}$$

Mantissa

Base

Esponente



Lo standard IEEE 754

I primi calcolatori utilizzavano ciascuno un differente sistema di convenzioni per rappresentare i floating point. Nel 1985 è stato definito uno standard che è oggi largamente accettato.

$$N = (-1)^{S} \cdot 1.F \cdot 2^{E-B}$$

Bit di segno

Mantissa frazionaria normalizzata

Esponente con bias

Esempio

Codifichiamo il numero -5.75 in singola precisione secondo lo standard IEEE 754.

Dobbiamo determinare segno, esponente e mantissa.

Poiché si tratta di un numero negativo il primo bit vale 1

Per determinare la mantissa scriviamo il numero in binario $101.11 = 1.0111 \times 2^2$. La mantissa è la parte a destra della virgola riempita con un opportuno numero di zeri. L'esponente vale 2, ma dobbiamo tenere conto del bias e quindi 2 → 129 che espresso in binario diventa 10000001

1 bit 8 bit 23 bit

S	Exp	Mantissa normalizzata
1	10000001	01110000000000000000000

Float, double, long double

L'esempio precedente si riferisce a un numero di tipo float o *singola precisione.*

Lo standard ANSI/IEEE754 definisce anche il numero di bit per ciascun campo per rappresentare numeri di tipo double (*doppia precisione*) e long double (*quadrupla precisione*).

Float

Double

Long double

Precisione e bit

- Cosa cambia tra i 3 tipi ? Ovviamente posso rappresentare numeri sempre più grandi ma è solo questo?
- In realtà la differenza sostanziale sta nella mantissa F.
- Questa infatti ci dà il numero di cifre significative che possiamo contare

$$F\ (23\ bit) = 2^{23} = 8388607$$

- significa che posso suddividere l'unità in 2^{23} intervalli e quindi ho una precisione

Float	$1/2^{23} = 1{,}2 \times 10^{-7}$
Double	$1/2^{52} = 2{,}2 \times 10^{-16}$
Long Double	$1/2^{112} = 1{,}2 \times 10^{-35}$

Overflow e underflow

Nelle operazioni fra floating point, possono verificarsi due condizioni di errore :

- Overflow: in un qualsiasi passo l'esponente calcolato è superiore al massimo rappresentabile; può verificarsi quando si maneggiano numeri dal valore assoluto molto elevato.

- Underflow: in un qualsiasi passo l'esponente calcolato è inferiore al minimo rappresentabile; può verificarsi quando si maneggiano numeri molto vicini allo "zero" della macchina.

Propagazione degli errori

I calcoli in virgola mobile vengono effettuati riportando i numeri allo stesso esponente, lavorando poi sulle mantisse e infine normalizzando il risultato.

Tali procedure introducono un arrotondamento nei calcoli:

la cifra meno significativa della mantissa sarà sempre affetta da un errore.

Codifica dei caratteri

Mentre per codificare numeri si usano tecniche basate sul loro valore, per codificare dei caratteri c'e' bisogno di una relazione convenzionale, ovvero di una tabella che faccia corrispondere a una data sequenza di bit un dato carattere.

Nel progettare tale tabella bisogna tener conto ovviamente anche di caratteri non direttamente stampabili, ma che rappresentino la formattazione del testo, come ad esempio il *carriage return* CR ("a capo").

Il codice ASCII

Lo standard internazionalmente adottato per la codifica dei caratteri è il codice ASCII, acronimo di American Standard Code for Information Interchange.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Il codice viene spesso esteso a 8 bit per comprendere set di caratteri specifici di una certa area geografica, come è,Ç

Linguaggio C

- Variabili e tipi

Variabili

Ogni variabile corrisponde a una determinata cella di memoria utilizzata per contenere un valore che può essere modificato dal programma.

In C i nomi delle variabili devono rispettare alcune regole:

- i nomi possono contenere lettere, numeri e il carattere _ (underscore)
- il primo carattere deve essere una lettera
- il C è case-sensistive `deviazione` ≠ `Deviazione`
- C keywords non possono essere utilizzate

Variabili

`Percent`	SI
`Y2x5_fg7h`	SI
`_1990_tax`	SI ma sconsigliato
`Ore#lavorate`	NO
`double`	NO
`Guadagno_annuale`	SI

Esistono vari modi per creare nomi di variabili a partire da più parole

`guadagno_annuale` `GuadagnoAnnuale`

DO & DON'T

Utilizzare nomi di variabili "descrittivi"

Adottare uno "stile" per dare i nomi alle variabili

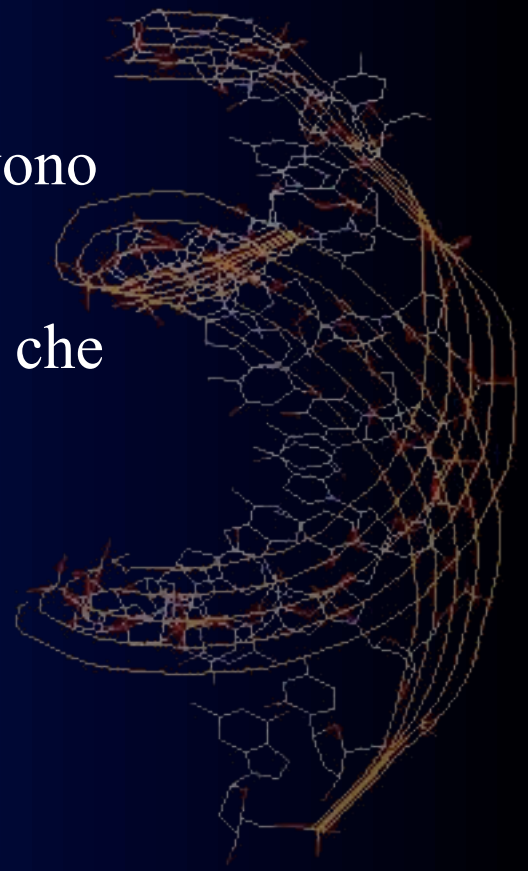
Non utilizzare il carattere underscore come primo carattere di un nome se non è indispensabile

Non utilizzare nomi di variabili composti da tutte lettere maiuscole

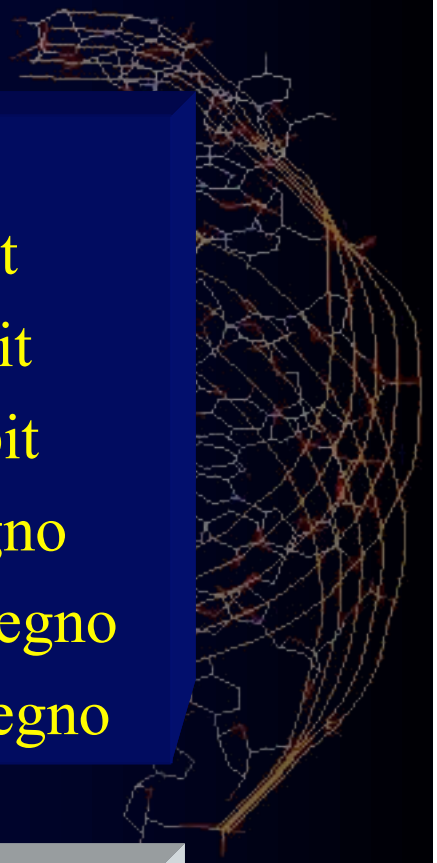
Variabili e tipi

Tutte le variabili C prima di essere utilizzate devono essere dichiarate.

La dichiarazione serve per stabilire il tipo di dati che la variabile può contenere



Tipi semplici predefiniti in C



Tipi interi:

`int` = intero con segno, 32 bit

`short` = intero con segno, 16 bit

`long` = intero con segno, 64 bit

`unsigned int` = come int ma senza segno

`unsigned short` = come short ma senza segno

`unsigned long` = come long ma senza segno

I tipi character sono in realtà interi di 8 bit:

`char` = intero di 1 byte

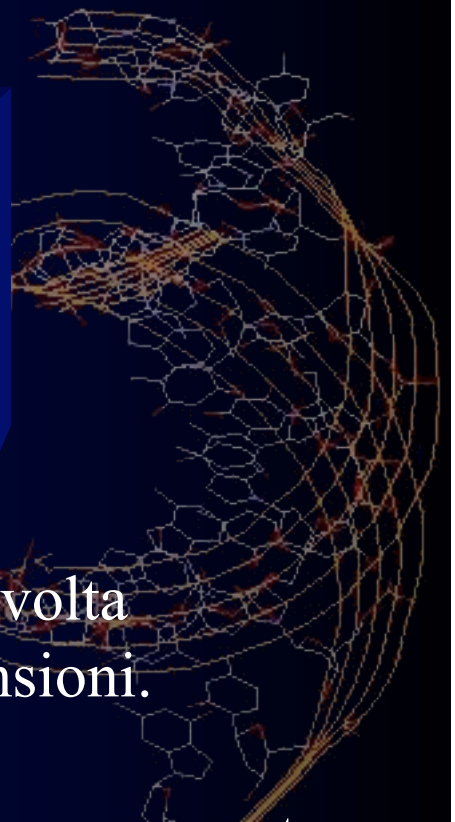
Tipi semplici predefiniti in C

Tipi float:

`float` = virgola mobile, 32 bit

`double` = virgola mobile, 64 bit

`long double` = virgola mobile, 128 bit



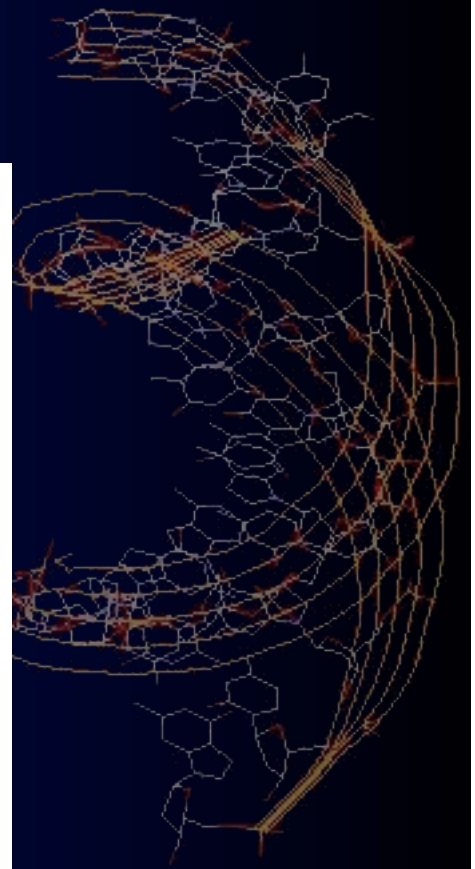
Il tipo più usato è il double, float è conveniente talvolta per risparmiare memoria in vettori di grandi dimensioni.

In aggiunta a questi tipi c'è il tipo speciale *void* che rappresenta l'assenza di valore (valore vuoto).

Vedremo meglio il suo utilizzo in seguito.

Esempio

```c
/* sizeof.c--Program to tell the size of the C variable */
/* type in bytes */

#include <stdio.h>

int main()
{

printf( "\nA char is %d bytes", sizeof( char ));
printf( "\nAn int is %d bytes", sizeof( int ));
printf( "\nA short is %d bytes", sizeof( short ));
printf( "\nA long is %d bytes", sizeof( long ));
printf( "\nAn unsigned char is %d bytes", sizeof( unsigned char ));
printf( "\nAn unsigned int is %d bytes", sizeof( unsigned int ));
printf( "\nAn unsigned short is %d bytes", sizeof( unsigned short ));
printf( "\nAn unsigned long is %d bytes", sizeof( unsigned long ));
printf( "\nA float is %d bytes", sizeof( float ));
printf( "\nA double is %d bytes\n", sizeof( double ));
 return 0;
}
```



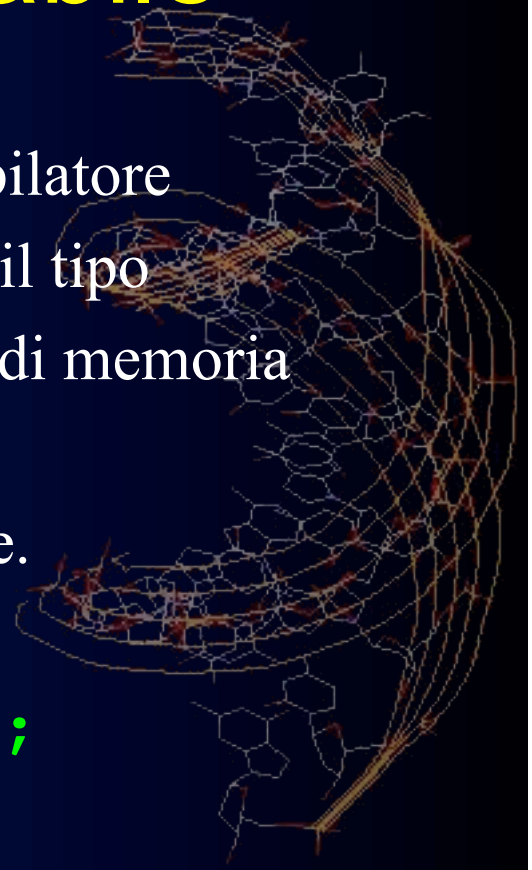
Inizializzare una variabile

Quando si dichiara il tipo di una variabile il compilatore riserva una porzione di memoria atta a contenere il tipo di variabile richiesto. Il contenuto di queste celle di memoria non è però definito.

E' quindi consigliabile inizializzare ogni variabile.

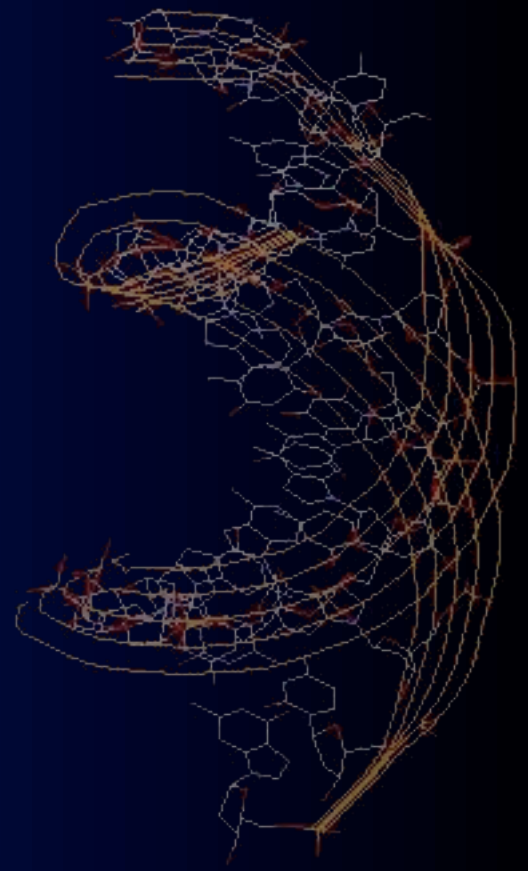
```
…
int count;
count = 0;
….
```

```
……
int count = 0;
…..
```

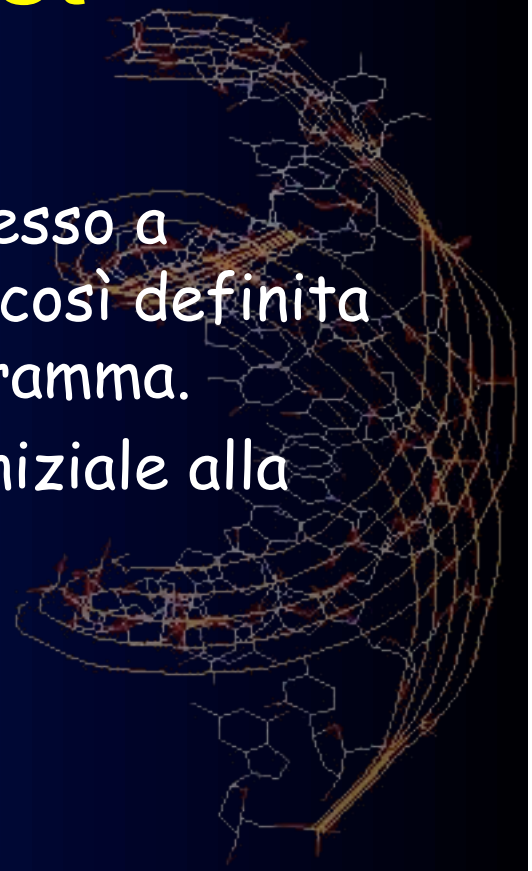


Esempio

```c
/* STAMPA2: stampa il valore di una variabile intera */


#include <stdio.h>

main()
{
    int anno;       /*dichiara che puï contenere un intero <1>*/

    printf("\n  Stampa il valore di una variabile numerica:\n\n");

    anno = 1993;                                    /*assegna valore*/
    printf("Mercato comune: %d\n",anno);            /*e lo stampa <2>*/
}

/* Note:
    <1> Dopo questa dichiarazione, la variabile anno puï essere usata
        liberamente. Il compilatore controlla comunque che non si cerchi
        di mettervi un dato del tipo sbagliato (cioä non un int).
    <2> La printf stampa come al solito la stringa che le viene passata,
        ma la specifica di stampa %d viene sostituita con il valore della
        variabile passata come secondo argomento (anno).
*/
```



Il qualificatore const



Il qualificatore **const** può essere premesso a qualsiasi tipo, indicando che la variabile così definita non verrà modificata nel corso del programma.

In tal caso bisogna assegnare il valore iniziale alla variabile in fase di definizione:

```
const int max_iterazioni = 100;
const char no = 'n';
const float pigreco = 3.141592;
```

La direttiva `#define`

```
#define CONSTNAME value
```

Crea una costante di nome CONSTNAME di valore value

Esempio:

```
#define PI 3.14159
```

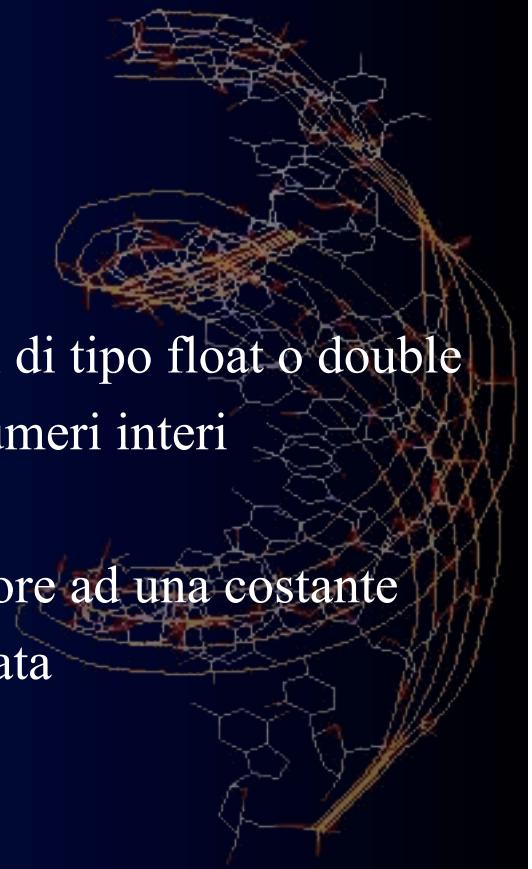
Questa direttiva può essere posta in ogni punto del codice

DO & DON'T

Comprendere il numero di bytes
che il proprio computer utilizza
per i vari tipi di variabili

Inizializzare le variabili

Utilizzare le "costanti" per rendere
i programmi più leggibili



Non utilizzare varibili di tipo float o double
per immagazzinare numeri interi

Non assegnare un valore ad una costante
dopo averla inizializzata

```c
/* Demonstrates variables and constants */
#include <stdio.h>

/* Define a constant to convert from pounds to grams */
#define GRAMS_PER_POUND 454

/* Define a constant for the start of the next century */
const int TARGET_YEAR = 2020;

 /* Declare the needed variables */
long weight_in_grams, weight_in_pounds;
int year_of_birth, age_in_2010;

 int main()
 {
 /* Input data from user */

 printf("Enter your weight in pounds: ");
 scanf("%d", &weight_in_pounds);
 printf("Enter your year of birth: ");
 scanf("%d", &year_of_birth);

 /* Perform conversions */

 weight_in_grams = weight_in_pounds * GRAMS_PER_POUND;
 age_in_2020 = TARGET_YEAR - year_of_birth;

 /* Display results on the screen */

 printf("\nYour weight in grams = %ld", weight_in_grams);
 printf("\nIn 2020 you will be %d years old\n", age_in_2020);

 return 0;
 }
```