Argomenti del corso

Rappresentazione dell'informazione.
Architettura del calcolatore. Linguaggio Assembler

Concetti di base della programmazione
Concetto di algoritmo. Il calcolatore come esecutore di algoritmi. Linguaggi di programmazione ad alto livello. Sintassi e semantica di un linguaggio di programmazione. Metodologie di programmazione strutturata. Principi fondamentali di progetto e sviluppo di semplici algoritmi.

Programmare in C
Tipi fondamentali. Istruzioni semplici, strutturate e di salto. Funzioni. Ricorsione. Riferimenti e puntatori. Array. Strutture e unioni. Memoria libera. Visibilità e collegamento. Algoritmi di ricerca e di ordinamento.

Concetti di base della programmazione a oggetti
Limitazioni dei tipi derivati. Il concetto di tipo di dato astratto.

Programmare in C++
Classi. Operatori con oggetti classe. Altre proprietà delle classi. Classi per l'ingresso e per l'uscita.

Progettare ed implementare tipi di dato astratti
Alcuni tipi di dato comuni con le classi: Liste, Code, Pile.

Rappresentazione dell'Informazione

L'informazione è qualcosa di astratto.
Per poterla manipolare bisogna rappresentarla.

In un calcolatore i vari tipi di informazione (testi, figure, numeri, musica,…) si rappresentano per mezzo di sequenze di bit (cifre binarie).

Bit è l'abbreviazione di Binary digIT, numero binario.

Il bit è l'unità di misura elementare dell'informazione, ma anche la base del sistema numerico utilizzato dai computer. Può assumere soltanto due valori: 0 o 1.

Byte è l'unità di misura dell'informazione che corrisponde ad 8 bit.

Rappresentazione dell'Informazione

Quanta informazione può essere contenuta in una sequenza di n bit?
L'informazione corrisponde a tutte le possibili disposizioni con ripetizione di due oggetti (0 ed 1) in n caselle (gli n bit), ossia $\mathbf{2^n}$

Esempio: n=2.
00
01
10
11

ATTENZIONE: Una stessa sequenza di bit può rappresentare informazione differente.

Per esempio 01000001 rappresenta
- l'intero 65
- il carattere 'A'
- il colore di un puntino sullo schermo

testo
disegni
immagini
numeri
musica
...

dispositivi di conversione

IN

OUT

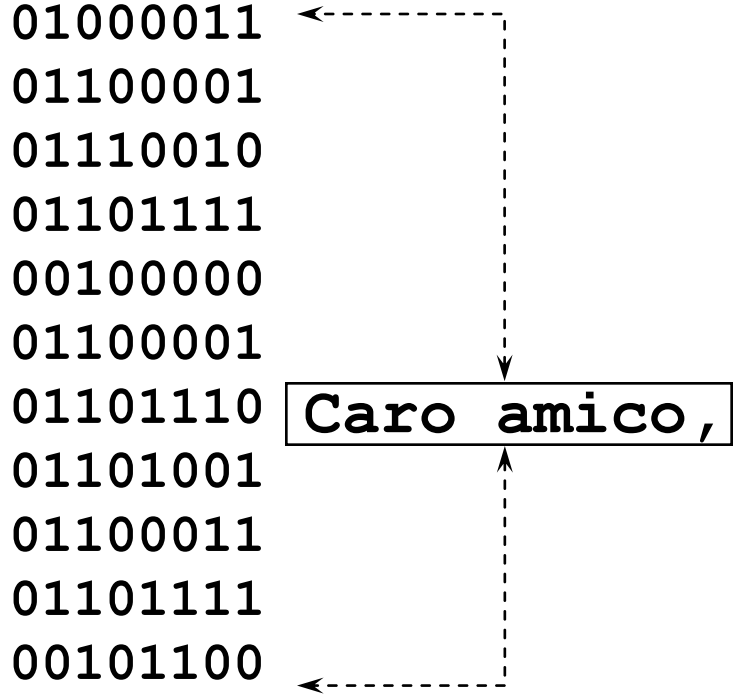
sequenze
di bit

Calcolatore

Rappresentazione del testo (1)

Codifica ASCII (American Standard Code for Information Interchange)
Standard su 7 bit (il primo bit del byte sempre 0)

Sequenze	Caratteri
00110000	0
00110001	1
00110010	2
...	...
00111001	9
...	...
01000001	A
01000010	B
01000011	C
...	...
01100001	a
01100010	b
...	...



Codifica ASCII – Tabella di corrispondenza

3 cifre più significative

000	001	010	011	100	101	110	111	
NUL	DLE	SP	0	@	P	`	p	0000
SOH	XON	!	1	A	Q	a	q	0001
STX	DC2	"	2	B	R	b	r	0010
ETX	XOFF	#	3	C	S	c	s	0011
EQT	DC4	$	4	D	T	d	t	0100
ENQ	NAK	%	5	E	U	e	u	0101
ACK	SYN	&	6	F	V	f	v	0110
BEL	ETB	'	7	G	W	g	w	0111
BS	CAN	(	8	H	X	h	x	1000
HT	EM	)	9	I	Y	i	y	1001
LF	SUB	*	:	J	Z	j	z	1010
VF	ESC	+	;	K	[	k	{	1011
FF	FS	,	<	L	\	l	|	1100
CR	GS	-	=	M	]	m	}	1101
SO	RS	.	>	N	^	n	~	1110
SI	US	/	?	O	_	o	DEL	1111

4 cifre meno significative

Rappresentazione dei numeri naturali (1)

Base dieci

- Cifre: 0, 1, 2, 3, 4, 6, 7, 8, 9
- Rappresentazione posizionale

$(123)_{dieci}$ *significa* $1\times10^2 + 2\times10^1 + 3\times10^0$

Base due

- Cifre: 0, 1
- Rappresentazione posizionale

$(11001)_{due}$ *significa* $1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0\ (= 25)_{dieci}$

Rappresentazione dei numeri naturali (2)

Data una base $\beta \geq$ **due**

Ogni numero naturale *N* minore di β ($N < \beta$) è associato ad un simbolo elementare detto **cifra**

BASE	CIFRE
due	0 1
cinque	0 1 2 3 4
otto	0 1 2 3 4 5 6 7
sedici	0 1 2 3 4 5 6 7 8 9 A B C D E F

Rappresentazione dei numeri naturali (3)

I numeri naturali maggiori o uguali a β possono essere rappresentati da una sequenza di cifre secondo la *rappresentazione posizionale*

Se un numero naturale $N \geq \beta$ è rappresentato in base β dalla sequenza di cifre:

$$a_{p-1}a_{p-2}\ldots a_1 a_0$$

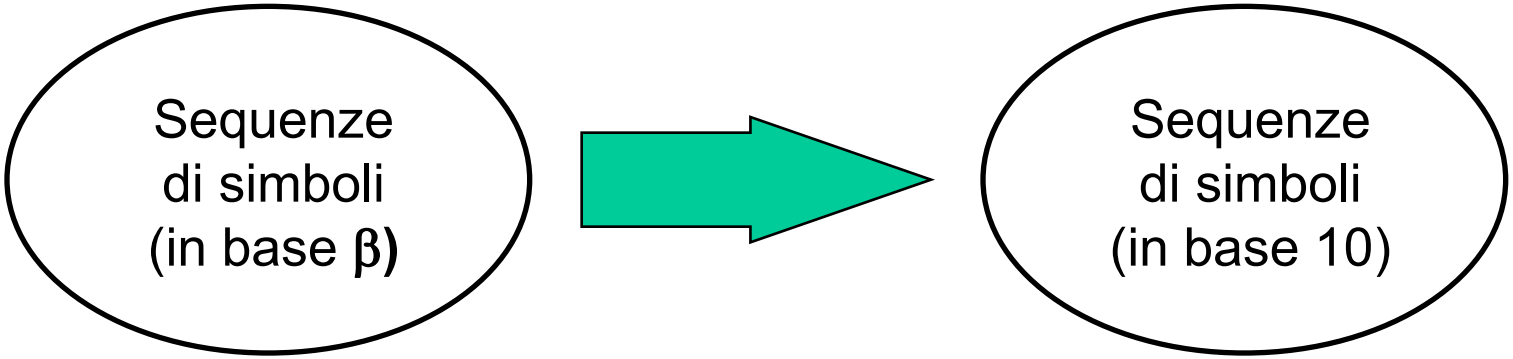
allora *N* può essere espresso come segue:

$$N = \sum_{i=0}^{p-1} a_i \beta^i = a_{p-1}\beta^{p-1} + a_{p-2}\beta^{p-2} + \ldots + a_2\beta^2 + a_1\beta + a_0$$

dove a_0 è detta «**cifra meno significativa»** e a_{p-1} **«cifra più significativa»**

Rappresentazione dei numeri naturali (4)

Data una sequenza di cifre in base β, a quale numero naturale corrisponde?



$a_{p-1}a_{p-2}\ldots a_1a_0$ $\quad N\,?$

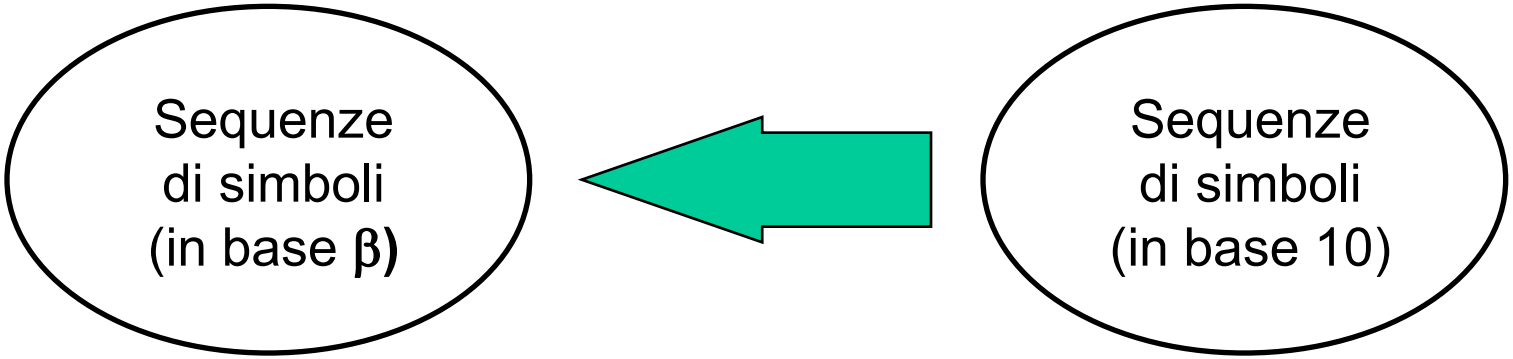
656_8 $\quad 6\cdot\boldsymbol{8}^2+5\cdot\boldsymbol{8}^1+6\cdot\boldsymbol{8}^0 \quad 430$

$A03_{16}$ $\quad 10\cdot\boldsymbol{16}^2+0\cdot\boldsymbol{16}^1+3\cdot\boldsymbol{16}^0 \quad 2563$

1101_2 $\quad 1\cdot\boldsymbol{2}^3+1\cdot\boldsymbol{2}^2+0\cdot\boldsymbol{2}^1+1\cdot\boldsymbol{2}^0 \quad 13$

Rappresentazione dei numeri naturali (5)

Data la base β ed un numero naturale N, trovare la sequenza di cifre che rappresenta N in base β



$a_{p-1}a_{p-2}\ldots a_1a_0$ N

Esempio: da base 10 a base 2

$N = 23$

inizio

	QUOZIENTE	RESTO	Rappresentazione
	23	-	
div 2	11	1	11*2+1
div 2	5	1	((5*2)+1)*2+1
div 2	2	1	((((2*2)+1)*2)+1)*2+1
div 2	1	0	(((((1*2)+0)*2)+1)*2)+1)*2+1
div 2	**0**	1	((((((0*2+1)*2+0)*2+1)*2+1)*2)+1

fine

$(10111)_{due}$

$a_4 a_3 a_2 a_1 a_0$

che in base dieci vale $1*2^4+\ldots+1=(23)_{dieci}$, cvd

Rappresentazione dei numeri naturali (7)

Sia **mod** il resto e **div** il quoziente della divisione intera

Esempio:
23 **div** 2 = 11
23 **mod** 2 = 1

Procedimento mod/div

Se $N = 0$ porre $a_0 = 0$; => fine

Altrimenti: porre $q_0 = N$ e poi eseguire la seguente procedura iterativa:

$$q_1 = q_0 \text{ div } \beta \qquad a_0 = q_0 \text{ mod } \beta$$
$$q_2 = q_1 \text{ div } \beta \qquad a_1 = q_1 \text{ mod } \beta$$
$$\ldots$$
$$q_{p-1} = q_{p-2} \text{ div } \beta \qquad a_{p-1} = q_{p-2} \text{ mod } \beta$$

NB: Il risultato della **mod** è sempre una cifra valida in base β, perché restituisce sempre un numero fra 0 e β-1 (estremi inclusi).

fino a che q_p è uguale a 0;

Il procedimento si arresta quando $q_p = 0$ (più precisamente subito dopo aver calcolato a_{p-1}). p è proprio il numero di cifre necessario per rappresentare N in base β

Rappresentazione dei numeri naturali (8)

Inizio $q_0 = N$

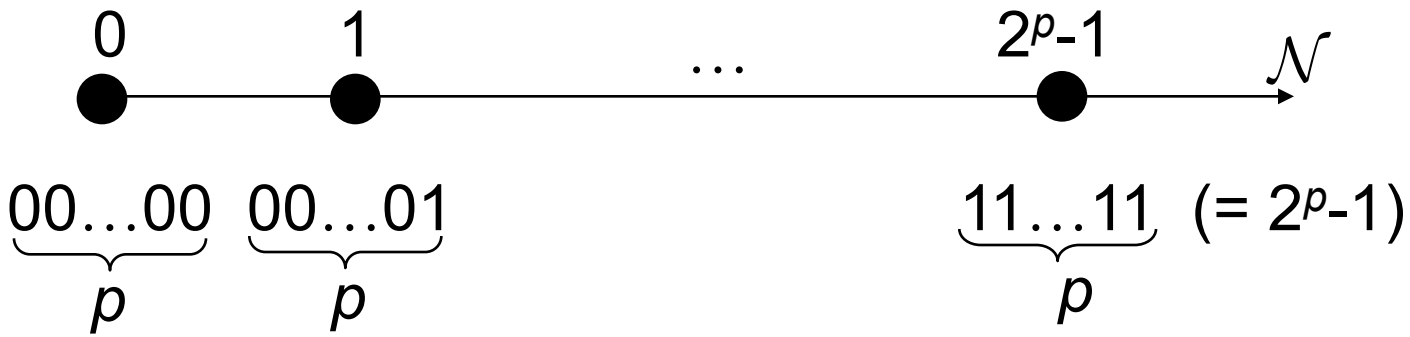
	QUOZIENTE	RESTO
	$q_0 = N$	-
div β	$q_1 = q_0 / \beta$	a_0
div β	$q_2 = q_1 / \beta$	a_1
div β	$q_3 = q_2 / \beta$	a_2
div β	$q_4 = q_3 / \beta$	a_3
div β	$q_5 = q_4 / \beta$	a_4
div β	$q_6 = q_5 / \beta$	a_5
	$q_7 = \mathbf{0}$	a_6

fine

$$N = a_6 \cdot \beta^6 + a_5 \cdot \beta^5 + a_4 \cdot \beta^4 + a_3 \cdot \beta^3 + a_2 \cdot \beta^2 + a_1 \cdot \beta^1 + a_0 \cdot \beta^0$$

Potenza della base: $2^p \leftrightarrow (1\underbrace{00\ldots00}_{p})_{due}$

Intervallo di rappresentabilità con p bit



p	**Intervallo** $[0, 2^p-1]$
8	[0, 255]
16	[0, 65535]
32	[0, 4294967295]

NB: per la generica base β, l'intervallo di rappresentabilità con p cifre è $[0, \beta^p-1]$

Rappresentazione dei numeri naturali (10)

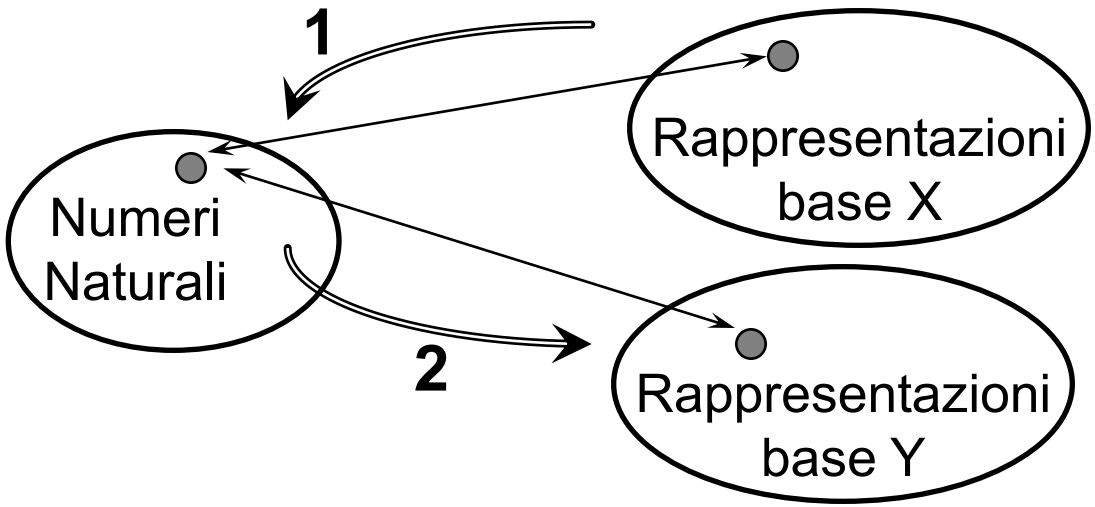
Calcolatore lavora con un numero finito di bit

- Supponiamo che p = 16 bit
- A = 0111011011010101 (30421)
  B = 1010100001110111 (43127)
- Poiché A + B (73552) è maggiore di 2^p-1(65535), quindi non è rappresentabile su p bit, si dice che la somma ha dato luogo ad ***overflow***
- In generale, ci vogliono p+1 bit per la somma di due numeri di p bit

A = 1111111111111111 (65535)
B = 1111111111111111 (65535)

1111111111111110 (131070)

Numeri naturali - Cambio di base (1)



Da base X a base Y

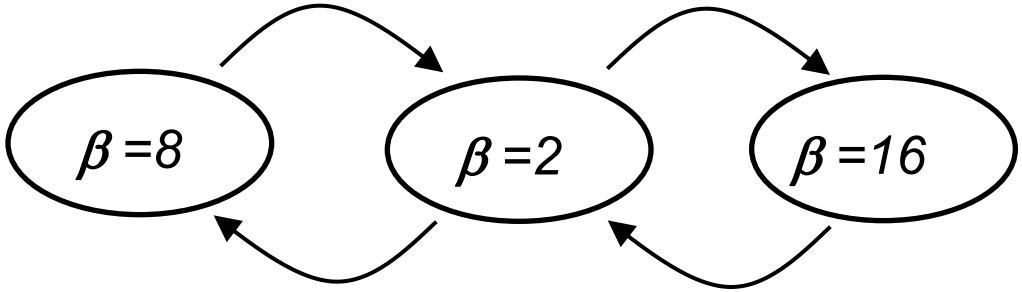
Trovare la rappresentazione in base 9 di $(1023)_{cinque}$

1) Trasformo in base 10 ed ottengo 138
2) Applico il procedimento `mod/div` ed ottengo $(163)_{nove}$

Casi particolari (potenze di 2)

$\beta = 8$

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



$\beta = 16$

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

$(657)_{otto}$ → $(110\ 101\ 111)_{due}$

$(A03)_{sedici}$ → $(1010\ 0000\ 0011)_{due}$

$(FFE4)_{sedici}$ → $(1111\ 1111\ 1110\ 0100)_{due}$

Prima rappresentazione: *rappresentazione in modulo e segno*

Trasformazione a => A

Sia **a** *(es.* **a**=+*3,* **a**=-*3, …)* il numero intero che si vuole rappresentare ***in modulo e segno*** su su *p* bit. La rappresentazione **A** (es. 0011, 1011, ...) di **a** è data da:

$$\mathbf{A} = a_{p-1},\ldots,a_0 = (segno\_a, ABS\_a)$$

dove *ABS_a* è la rappresentazione (come numero naturale) del valore assoluto di **a** su *p*-1 bit (in particolare *ABS_a* è rappresentato mediante i bit $a_{p-2},\ldots,a_0$),
e *segno_a* è un unico bit che vale 0 se **a** >= 0 e 1 se **a** <= 0).
NB: ad **a**=0 corrispondono due rappresentazioni: +*zero* (0,00..0) e -*zero* (1,00..0)

Ad esempio, per *p* = 4 si ha:

a=+3 => *segno_a* = 0, *ABS_a* = 011 (naturale 3 rappresentato su 3 cifre) => **A**=0011
a= -3 => *segno_a* = 1, *ABS_a* = 011 (naturale 3 rappresentato su 3 cifre) => **A**=1011

Prima rappresentazione: *rappresentazione in modulo e segno*

Trasformazione A => a

Data una rappresentazione **A** di un intero **in modulo e segno** su *p* bit, come si risale all'intero **a** da esso rappresentato?
La rappresentazione **A** va vista come composta di due parti: **A**=($\underbrace{a_{p-1}}$, $\underbrace{a_{p-2}\ldots a_0}$)
dopodiché:

$$\mathbf{a} = (a_{p-1} == 0)\ ?\ +ABS\_a\ :\ -ABS\_a$$

dove *ABS_a* è il naturale corrispondente ad $a_{p-2}\ldots a_0$

Ad esempio, per *p* = 4 si ha:

A=0011 => viene visto come (0,011) => **a** = +011 (+*tre*)
A=1011 => viene visto come (1,011) => **a** = -011 (-*tre*)

NB: 0000 rappresenta +*zero*, mentre 1000 rappresenta –*zero*.

A	$\{0,1\}^4$	a
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	-0
9	1001	-1
10	1010	-2
11	1011	-3
12	1100	-4
13	1101	-5
14	1110	-6
15	1111	-7

Rappresentazione di interi in modulo e segno su calcolatore con p=4 bit

numero negativo ⇔ bit più significativo della rappresentazione uguale a 1

(*zero* rappresentato due volte)

Intervallo di rappresentabilità in modulo e segno su *p* bit
(doppia rappresentazione dello *zero*)

$$[-(2^{p-1}-1),+2^{p-1}-1]$$

- $p = 4$ [-7,+7]
- $p = 8$ [-127,+127]
- $p = 16$ [-32767,+32767]

NB: prima di applicare l'algoritmo per **a**=>**A** occorre verificare che a sia rappresentabile su *p* bit, altrimenti l'algoritmo conduce a risultati sbagliati.
Ad esempio:
tentando di rappresentare **a**=-9 su *p*=4 bit, si ottiene:
A=(*segno_a*, *ABS_a*), dove *segno_a*=1 e *ABS_a* = 9
ma il naturale 9 (1001) non è rappresentabile su 3 bit!!

Seconda rappresentazione: *rappresentazione in complemento a 2*
Trasformazione a => A

Sia **a** *(es.* **a**=+*3,* **a**=-*3, …)* il numero intero che si vuole rappresentare ***in complemento a 2*** su *p* bit. La rappresentazione **A** (es. 0011, 1101, …) di **a** è data da:

$$\mathbf{A} = a_{p-1}\ldots a_0 = (\mathbf{a} >= 0)\ ?\ ABS\_a\ :\ (due^p - ABS\_a)$$

dove sia *ABS_a* che (*due*^{p}*-ABS_a*) sono rappresentati in base due come numeri naturali su *p* bit.

Ad esempio, per *p* = 4 si ha:

a = 0 => *ABS_a* = 0, e il naturale 0 ha rappr. 0000 su 4 bit => **A** = 0000
a = 1 => *ABS_a* = 1, e il naturale 1 ha rappr. 0001 su 4 bit => **A** = 0001
a = 7 => *ABS_a* = 7, e il naturale 7 ha rappr. 0111 su 4 bit => **A** = 0111
a = -1 => *ABS_a* = 1, 16-1=15, dove il naturale 15 ha rappr. 1111 su 4 bit => **A** = 1111
a = -2 => *ABS_a* = 2, 16-2=14, dove il naturale 14 ha rappr. 1110 su 4 bit => **A** = 1110
a = -8 => *ABS_a* = 8, 16-8=8, dove il naturale 8 ha rappr. 1000 su 4 bit => **A** = 1000

NB: La rappresentazione in complemento a 2 è anche detta in *complemento alla base*. Infatti lo stesso procedimento può essere generalizzato per rappresentare interi in basi diverse da due.

Seconda rappresentazione: *rappresentazione in complemento a 2*
Trasformazione A => a

Data una rappresentazione **A** = $a_{p-1}\ldots a_0$ di un intero **in complemento a due** su *p* bit, come si risale all'intero **a** da esso rappresentato?

$$\mathbf{a} = (a_{p-1} == 0)\ ?\ +\mathbf{A} : -(due^p - \mathbf{A})$$

dove sia **A** che (due^p-**A**) vengono visti come naturali su *p* bit.

Ad esempio, per *p* = 4 si ha:

A = 0000 => **a** = 0 (*zero*)
A = 0001 => **a** = +1 (+*uno*)
A = 0111 => **a** = +7 (+*sette*)
A = 1000 => 16-8 = 8 => **a** = -8 (-*otto*)
A = 1001 => 16-9 = 7 => **a** = -7 (-*sette*)
A = 1111 => 16-15=1 => **a** = -1 (-*uno*)

A	$\{0,1\}^4$	a
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	-7
10	1010	-6
11	1011	-5
12	1100	-4
13	1101	-3
14	1110	-2
15	1111	-1

Rappresentazione di interi in complemento a due su calcolatore con p=4 bit

Anche in questo caso:

numero negativo $\Leftrightarrow$ *bit più significativo della rappresentazione uguale a 1*

Inoltre, a differenza della rappresentazione in modulo e segno, non viene sprecata nessuna rappresentazione (lo *zero* è rappresentato una volta sola)

Intervallo di rappresentabilità in complemento a 2 su *p* bit

$$[-2^{p-1},+2^{p-1}-1]$$

- $p = 4$ [-8,+7]
- $p = 8$ [-128,+127]
- $p = 16$ [-32768,+32767]

NB: prima di applicare l'algoritmo per a=>A occorre verificare che a sia rappresentabile su *p* bit, altrimenti l'algoritmo conduce a risultati sbagliati.
Ad esempio, tentando di rappresentare a=-9 su *p*=4 bit, si ottiene:
A=(due^4-9)=16-9 = 7 =>0111,
ma è sbagliato! Infatti -9 non è rappresentabile su 4 bit, ne servono almeno 5!
Che il risultato fosse sbagliato si poteva dedurre dal fatto che la rappresentazione del negativo -9 iniziava per 0 (0111 è infatti la rappresentazione di a=+7 su 4 bit!).

(su 5 bit, a=-9 ha rappresentazione (due^5-9)=23 => A=10111)

Numeri Interi (9/14)

Terza Rappresentazione: *rappresentazione con bias*
Trasformazione a => A

Sia **a** *(es.* **a**=+*3,* **a**=-*3, …)* il numero intero che si vuole rappresentare ***in rappresentazione con bias*** su *p* bit. La rappresentazione **A** (es. 1010, 0100, …) di **a** è data da:

$$\mathbf{A} = a_{p-1}\ldots a_0 = \mathbf{a}+(2^{p-1} - 1)$$

dove **a**+(2^{p-1} - 1) è supposto essere non negativo e dunque viene rappresentato come un naturale su *p* bit. La quantità (2^{p-1} - 1) è detta *bias*.

Ad esempio, per *p* = 4 si ha:

a = 0 => 0+(2^{4-1}-1) = 7, e il naturale 7 ha rappr. 0111 su 4 bit => **A** = 0111
a = 1 => 1+(2^{4-1}-1) = 8, e il naturale 8 ha rappr. 1000 su 4 bit => **A** = 1000
a = 8 => 8+(2^{4-1}-1) = 15, e il naturale 15 ha rappr. 1111 su 4 bit => **A** = 1111
a = -1 => -1+(2^{4-1}-1) = 6, e il naturale 6 ha rappr. 0110 su 4 bit => **A** = 0110
a = -2 => -2+(2^{4-1}-1) = 5, e il naturale 5 ha rappr. 0101 su 4 bit => **A** = 0101
a = -7 => -7+(2^{4-1}-1) = 0, e il naturale 0 ha rappr. 0000 su 4 bit => **A** = 0000

NB1: questa rappresentazione è anche detta *rappresentazione con polarizzazione o rappresentazione con traslazione*
NB2: come si vedrà nelle prossime slides, questa rappresentazione viene utilizzata nella rappresentazione dei numeri reali in virgola mobile

Numeri Interi (10/14)

Terza Rappresentazione: *rappresentazione con bias*
Trasformazione A => a

Sia **A** *(es.* 1010, 0100, …*)* il numero intero che si vuole rappresentare ***in rappresentazione con bias*** su *p* bit. La rappresentazione **a** (*es.* **a**=+*3,* **a**=-*3, …*) di **a** è data da:

$$\mathbf{a} = \mathbf{A}-(2^{p-1} - 1)$$

dove A viene visto come numero naturale su *p* bit.

Ad esempio, per *p* = 4 si ha:

A = 0111 => $7-(2^{4-1}-1) = 0$ => **a** = 0
A = 1000 => $8-(2^{4-1}-1) = 1$ => **a** = 1
A = 1111 => $15-(2^{4-1}-1) = 8$ => **a** = 8
A = 0110 => $6-(2^{4-1}-1) = -1$ => **a** = -1
A = 0101 => $5-(2^{4-1}-1) = -2$ => **a** = -2
A = 0000 => $0-(2^{4-1}-1) = -7$ => **a** = -7

NB: Lo zero viene rappresentato una sola volta (come accade in compl. a 2).

Intervallo di rappresentabilità nella *rappresentazione con bias*

$$[-2^{(p-1)}-1, +2^{(p-1)}], \quad \text{ossia } [-bias, bias+1]$$

- $p = 4$ [-7,+8] (*bias*=7)
- $p = 5$ [-15,+16] (*bias*=15)
- $p = 7$ [-127,+128] (*bias*=127)
- $p = 10$ [-1023,+1024] (*bias*=1027)
- $p = 14$ [-16383,+16384] (*bias*=16383)

NB: prima di applicare l'algoritmo per a=>A occorre verificare che a sia rappresentabile su *p* bit, altrimenti l'algoritmo conduce a risultati sbagliati.

Ad esempio, tentando di rappresentare a=-9 su *p*=4 bit, si ottiene:
$A=-9+(2^{4-1}-1)=-9+7 = -2$, che non è un numero naturale!

Numeri Interi (12/14)

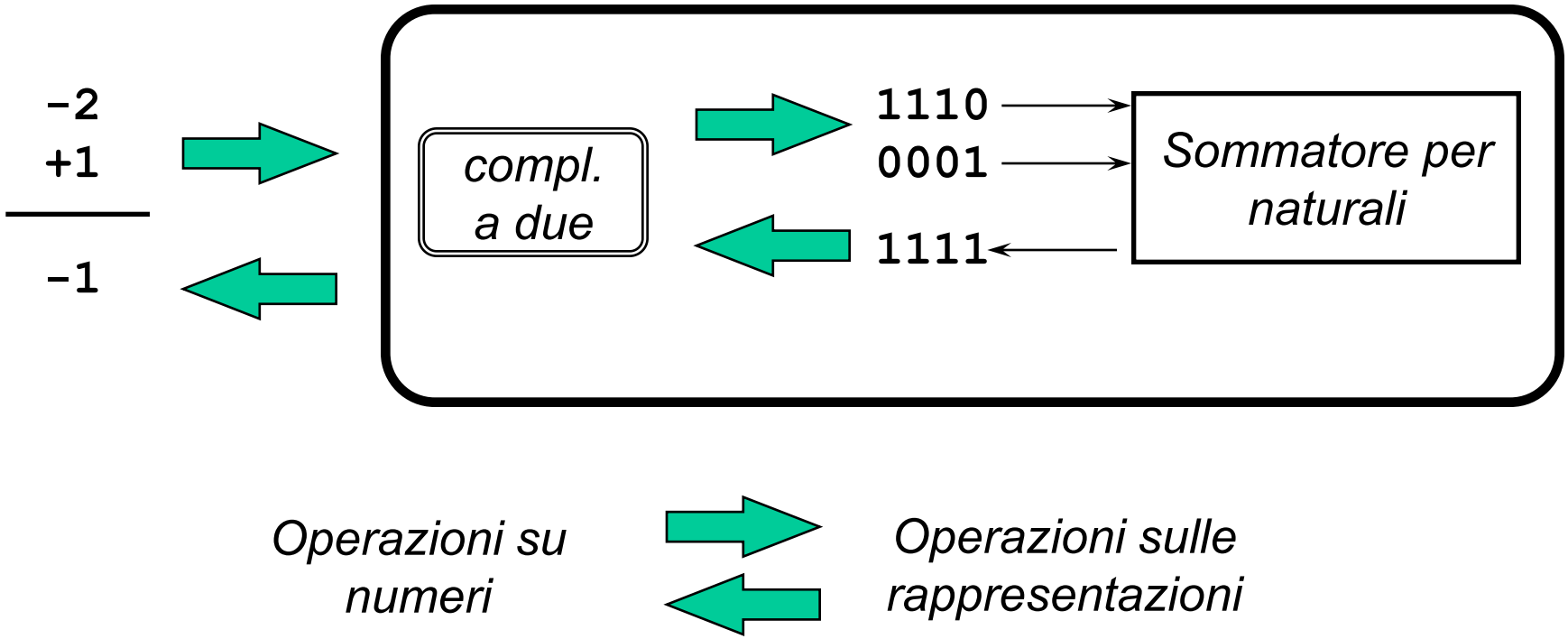
A	$\{0,1\}^5$	a^{ms}	a^{c2}	a^{bias}
0	00000	+0	0	-15
1	00001	+1	+1	-14
2	00010	+2	+2	-13
3	00011	+3	+3	-12
4	00100	+4	+4	-11
5	00101	+5	+5	-10
6	00110	+6	+6	-9
7	00111	+7	+7	-8
8	01000	+8	+8	-7
9	01001	+9	+9	-6
10	01010	+10	+10	-5
11	01011	+11	+11	-4
12	01100	+12	+12	-3
13	01101	+13	+13	-2
14	01110	+14	+14	-1
15	01111	+15	+15	0
16	10000	-0	-16	+1
17	10001	-1	-15	+2
18	10010	-2	-14	+3
19	10011	-3	-13	+4
20	10100	-4	-12	+5
21	10101	-5	-11	+6
22	10110	-6	-10	+7
23	10111	-7	-9	+8
24	11000	-8	-8	+9
25	11001	-9	-7	+10
26	11010	-10	-6	+11
27	11011	-11	-5	+12
28	11100	-12	-4	+13
29	11101	-13	-3	+14
30	11110	-14	-2	+15
31	11111	-15	-1	+16

Rappresentazioni degli interi su *p*=5 bit messe a confronto

Osservazione:

In complemento a due, **a**=-1 *è sempre rappresentazione dalla sequenza di p bit a 1 (11….11), qualunque sia il valore di p.*

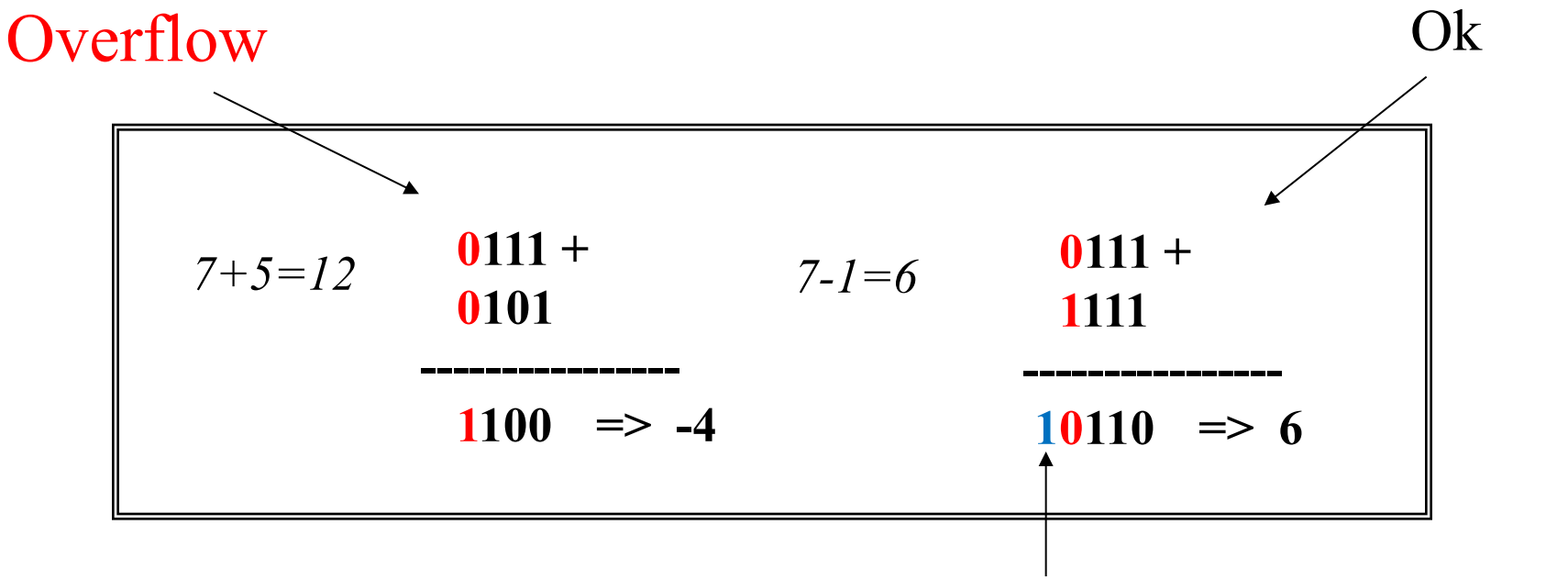
Numeri Interi (13/14)



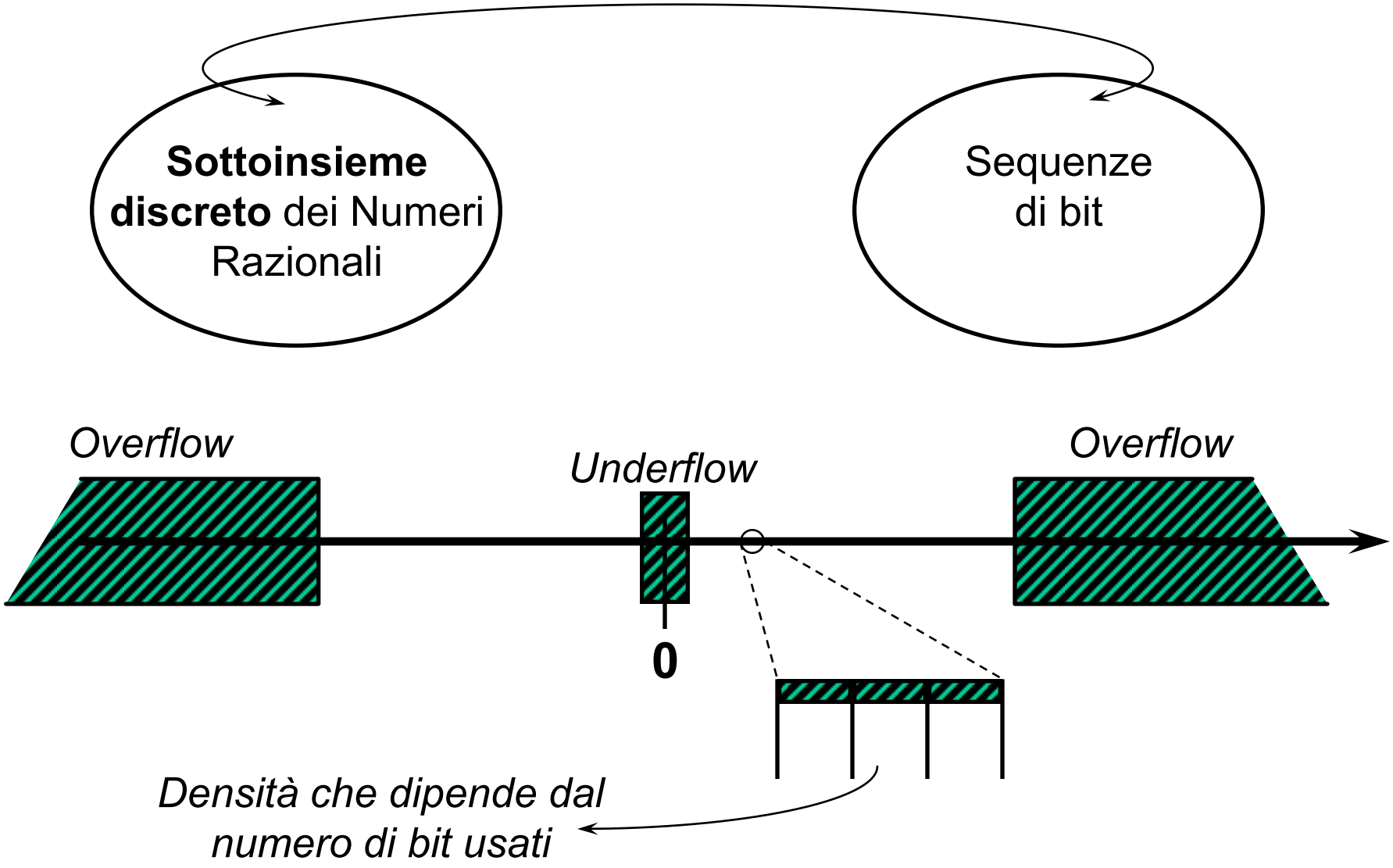
QUESTO è il motivo per cui i calcolatori rappresentano gli interi in complemento a due: ***non occorre una nuova circuiteria per sommare e sottrarre numeri interi, viene utilizzata la stessa dei numeri naturali!***

Numeri Interi (14/14)

Sommando due numeri interi si verifica un **overflow** quando i due numeri hanno lo stesso segno ed il risultato ha segno diverso



Numeri Reali



Numeri Reali – Virgola fissa (1/5)

- Si usa un numero fisso di bit per la parte intera ed un numero fisso di bit per la parte frazionaria.
- Sia *r* un numero reale da rappresentare. Di seguito, indichiamo con I(r) la parte intera e con F(r) la parte frazionaria di *r*
- Siano *p* i bit per rappresentare *r*: *f* per la parte frazionaria e (p-f) i bit la parte intera:

$$R = a_{p-f-1}\ldots a_0\, a_{-1}\ldots a_{-f} \qquad r \cong \sum_{i=-f}^{p-f-1} a_i \beta^i = a_{p-f-1}\beta^{p-f-1} + \ldots + a_0\beta^0 + \underbrace{a_{-1}\beta^{-1} + \ldots + a_{-f}\beta^{-f}}_{\text{parte frazionaria}}$$

Esempio: +1110.01 in base *due* vale +14.25 in base *dieci*

- NB: La virgola non si rappresenta
- La parte intera I(r) si rappresenta con le tecniche note
- Per la parte frazionaria si usa il procedimento seguente:

Numeri Reali – Virgola fissa(2/5)

Si usa la così detta procedura *parte frazionaria-parte intera:*

$f_0 = F(r)$
Se $f_0 \neq 0$ eseguire la seguente procedura iterativa:

$f_{-1} = F(f_0 * 2)$ $a_{-1} = I(f_0 * 2)$
$f_{-2} = F(f_{-1} * 2)$ $a_{-2} = I(f_{-1} * 2)$
...
fino a che f_{-j} è uguale a zero oppure si è raggiunta la precisione desiderata

Esempio:

***p* = 16 e *f* = 5**
r = +331,6875

Numeri Reali – Virgola fissa(3/5)

QUOZIENTE	RESTO
331	-
165	1
82	1
41	0
20	1
10	0
5	0
2	1
1	0
0	1

- **Dalla rappresentazione dei numeri naturali: 331 ⇔ 101001011;**

Numeri Reali – Virgola fissa(4/5)

F	I
f_{-1}=F(0.6875*2=**1.375**)=0.375	a_{-1}=I(1.375)=**1**
f_{-2}=F(0.375*2=**0.75**)=0.75	a_{-2}=I(0.75)=**0**
f_{-2}=F(0.75*2=**1.5**)=0.5	a_{-3}=I(1.5)=**1**
f_{-3}=F(0.5*2=**1.0**)=0	a_{-4}=I(1.0)=**1**

- r = +10100101110110 $\Rightarrow$ R = 0010100101110110
- Parte frazionaria ->

$$1*\ 2^{-1} + 0*2^{-2} + 1*2^{-3} + 1*2^{--4} \text{ -> } 0{,}5 + 0{,}125 + 0{,}0625 \text{ -> } 0{,}6875$$

- Dall'algoritmo precedente: 0,6875 $\Leftrightarrow$ 0,1011
- r = (+101001011,1011) base *due*

Numeri Reali – Virgola fissa(5/5)

ERRORE DI TRONCAMENTO

Rappresentare $r = 2.3$ con f = 6.

F	I
f_{-1}=F(0.3*2=**0.6**)=0.6	a_{-1}=I(**0.6**)=**0**
f_{-2}=F(0.6*2=**1.2**)=0.2	a_{-2}=I(1.2)=**1**
f_{-3}=F(0.2*2=**0.4**)=0.4	a_{-3}=I(0.4)=**0**
f_{-4}=F(0.4*2=**0.8**)=0.8	a_{-4}=I(0.8)=**0**
f_{-5}=F(0.8*2=**1.6**)=0.6	a_{-5}=I(0.6)=**0**
f_{-6}=F(0.6*2=**1.2**)=0.2	a_{-6}=I(0.8)=**1**

Per esprimere r sono necessarie un numero infinito di cifre!

(10.0 1001 1001 1001… = 10.01001 dove 1001 è la parte periodica)

Ne abbiamo a disposizione solo 6. Quindi si opera un *troncamento* alla 6ª cifra dopo la virgola.

Quindi, in realtà si rappresenta non r ma il numero r'

r' = 10.010011

$r' = 2 + 2^{-2} + 2^{-5} + 2^{-6} = 2 + 0.25 + 0.03125 + 0.015625 = 2.296875$

L'errore di troncamento è (2.3-2.296875)=0,00312499 ($< 2^{-6} = 0.015625$)

CALCOLO DI FISICA ASTRONOMICA

$m_e = 9 \times 10^{-28}$ gr; $m_{sole} = 2 \times 10^{+33}$ gr $\Rightarrow$ Intervallo di variazione $\approx 10^{+60}$.

Sarebbero quindi necessarie almeno 62 cifre, di cui 28 per la parte frazionaria.

Si hanno tante cifre perché l'intervallo da rappresentare è grande $\Rightarrow$ si separa l'intervallo di rappresentabilità dalla precisione, cioè dal numero di cifre.

Notazione Scientifica

$r = \pm m \cdot \beta^e$ m = mantissa; e = esponente

L'intervallo di rappresentabilità è fissato dal numero di cifre dell'esponente.

La precisione è fissata dal numero di cifre della mantissa.

Diverse rappresentazioni

Si fissa una forma standard

3.14 $0.314\ 10^{+1}$ $31.4\ 10^{-1}$

Numeri Reali – Virgola mobile(2/10)

Rappresentazione di un numero binario in virgola mobile:

+1110.01 numero reale binario in virgola fissa

esponenti espressi in binario

$+1.11001 \cdot due^{+11}$ … … $+111001 \cdot due^{-10}$

(alcune possibili rappresentazioni in virgola mobile)

Numeri reali *normalizzati*:

- mantissa con parte intera costituita da un solo bit di valore 1
- rappresentazione è una tripla costituita da tre numeri naturali

$$r \leftrightarrow R = \langle s, \quad E, \quad F \rangle$$

Standard IEEE 754-1985

La rappresentazione R è composta da tre naturali (s, E ed F), dove:

s = codifica del segno (1 bit)
F = codifica della parte frazionaria della mantissa su G bit
E = codifica dell'esponente su K bit

$$r = (s == 0)? \; [+(1+f) \cdot due^e)] : [-(1+f) \cdot due^e)]$$

$f = F/2^G$ è la parte frazionaria della mantissa (m=1+f= $1+F/2^G$)

$e = +E - (2^{K-1} - 1)$ è l'esponente rappresentato dal numero naturale E (*rappresentazione con polarizzazione*)

Conseguenza dell' "uno implicito" $\Rightarrow$ *lo zero non è rappresentabile!*

Half precision: 16 bit, $K = 5$ e $G = 10$. Esempi di trasformazione R => r

Esempio 1

R = {1,10011,1110100101} rappresenta il numero reale negativo con:

$f = F/2^G = F/2^{dieci} = 0.1110100101$

$e = E - (2^{K-1} - 1) = +10011 - (+01111) = +100$ (*bias=quindici*)

r = $-1.1110100101 \cdot due^{+quattro} = -11110.100101$

r = -30.578125 in base *dieci*

Esempio 2

R = {0,01111,0000000001} rappresenta il numero reale positivo:

$f = F/2^M = 1/2^G = due^{-dieci} = 0.0009765625$

$e = E - (2^{K-1} - 1) = 01111-01111 = 15-15=0$

r = $+1.0000000001 \cdot due^{+zero} = 2.0009765625 \cdot due^{+zero} = 2.0009765625$

r = + 2.0009765625 in base *dieci*

Half precision, esempi di trasformazione r => R

Esempio A – Rappresentare **r=2** *in half precision*
La rappresentazione di **r**= *2 è* R = {0,10000,0000000000}. Infatti, decodificando:
$f = F/2^G = 0$
$e = E - (2^{K-1} - 1) = 10000\text{-}01111 = 1$
r = +1.0000000000 · due^{+1} = 2 in base *dieci*

Esempio B – Rappresentare **r=0.5** *in half precision*
La rappresentazione di **r** = 0.5 *è* R = {0,01110,0000000000}. Infatti, decodificando:
$f = F/2^G = 0$
$e = E - (2^{K-1} - 1) = 01110\text{-}01111 = -1$
r = +1.0000000000 · due^{-1} = 0.5 in base *dieci*

Half precision, esempi di trasformazione r => R

Esempio C – Rappresentare **r=2.3** *in half precision*
Sapendo che **r** ha rappresentazione in virgola fissa 10.01001,
iniziamo a portarlo in forma normalizzata: 1.001001 $\cdot$ *due*$^{+1}$

Ora ci servono 10 cifre per la parte frazionaria della mantissa, le restanti verranno scartate, provocando il consueto *errore di troncamento*:

1.00 1001 1001 ~~*1001 1001*~~*...* $\cdot$ *due*$^{+1}$

F (prime 10 cifre a destra della virgola. Le altre vengono ignorate)

A questo punto la rappresentazione è immediata (per trovare *E* si faccia riferimento all'esempio A: **R**={0,10000,0010011001}

Esempio (numeri con modulo massimo, i cosiddetti ± ∞):

R={*0*,11111,1111111111} (massimo numero rappresentabile, +∞)
R={*1*,11111,1111111111} (minimo numero rappresentabile, -∞)

$f = F/2^{G} = F/2^{dieci} = 0.1111111111$
$e = + E -(2^{K-1} - 1) = +11111 - (+01111) = +10000$ (+*sedici*)

$|\mathbf{r}| = 1.1111111111 \cdot due^{+10000} < 2^{(2^{(K-1)}+1)} = \mathbf{2^{bias+2}} = 2^{17} = 131072$
$|\mathbf{r}| = 131008 \cong 1.3 \cdot 10^{+5}$ in base *dieci*

Riassumendo, nella rappresentazione *half precision*:
il «+∞» corrisponde a $\cong 2^{+17}$
il « -∞» corrisponde a $\cong -2^{+17}$

NB: L'intervallo di rappresentabilità [-∞, +∞] è approssimabile dal seguente: $[\cong -2^{bias+2}, \cong 2^{bias+2}]$

Esempio (numeri con modulo minimo, i cosiddetti ± 0):

R={*0*, 00000,0000000000} (minimo numero positivo, +0)
R= {*1*,00000,0000000000} (massimo numero negativo, -0)

$f = F/2^{G} = F/2^{dieci} = 0.0000000000$
$e = + E -(2^{K-1} - 1) = +00000 - (+01111) = -01111 = -quindici$

$|\mathbf{r}| = 1.0000000000 \cdot due^{-01111} = \mathbf{2^{-bias}}$
$|\mathbf{r}| = 2^{-15} \cong 0.31 \cdot 10^{-4}$

Riassumendo, nella rappresentazione *half precision*:
il «+0» corrisponde a $2^{-15} \cong 0.31 \cdot 10^{-4}$
il « -0» corrisponde a $-2^{-15} \cong -0.31 \cdot 10^{-4}$

Numeri Reali – Virgola mobile(9/10)

Standard IEEE 754-1985 prevede:

Rappresentazione in ***single precision*** su **32 bit** con ***K* = 8** e ***G* = 23**
Intervallo di rappresentabilità:

massimo modulo $\cong 2^{(bias+2)} = 2^{+129} \cong 6.8 \cdot 10^{+38}$
minimo modulo $= 2^{-bias} = 2^{-127} \cong 0.58 \cdot 10^{-38}$

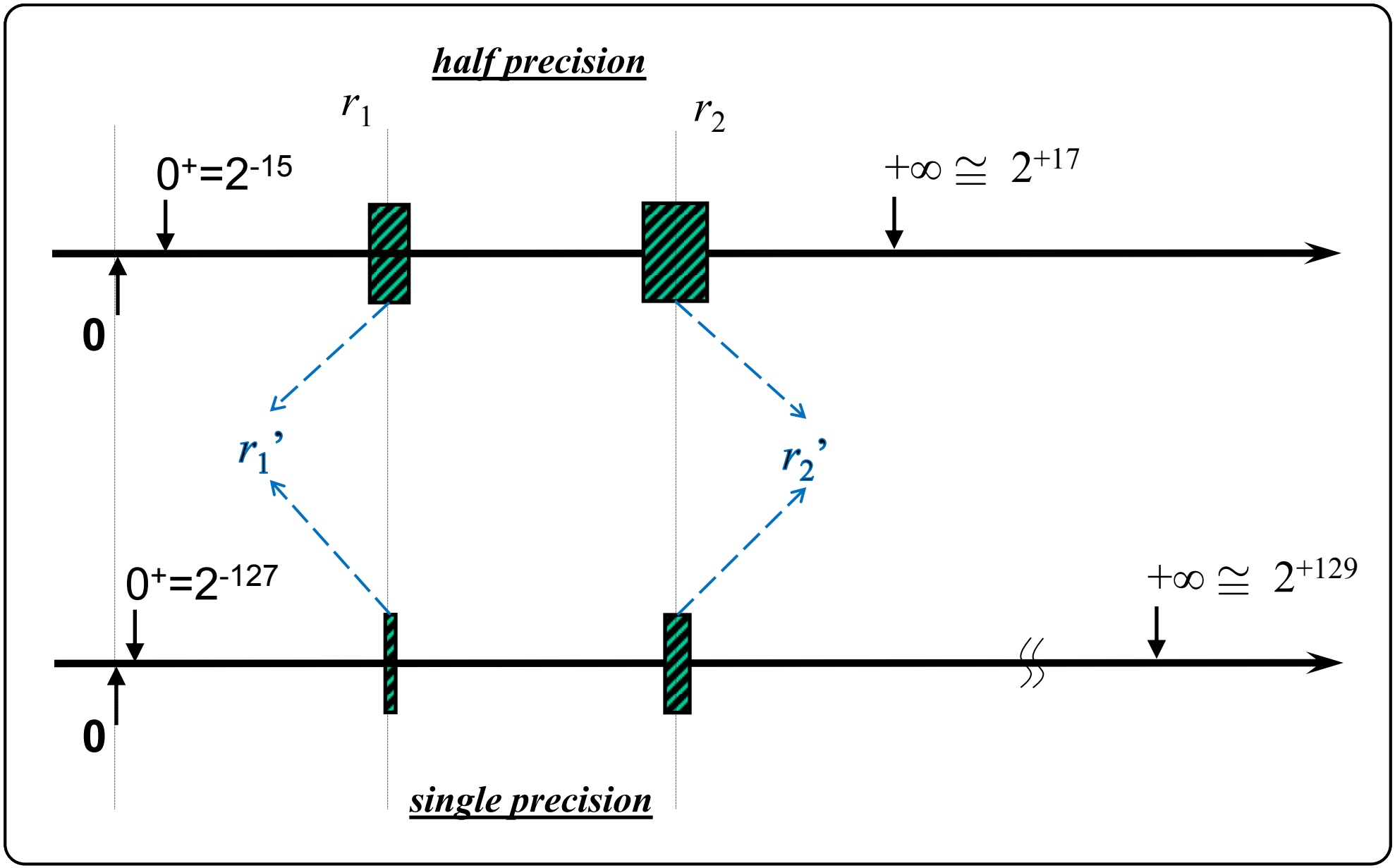
Rappresentazione in ***double precision*** su **64 bit** con ***K* = 11** e ***G* = 52**
Intervallo di rappresentabilità:

massimo modulo $\cong 2^{(bias+2)} = 2^{+1025} \cong 3.6 \cdot 10^{+308}$
minimo modulo $= 2^{-bias} = 2^{-1023} \cong 1.1 \cdot 10^{-308}$

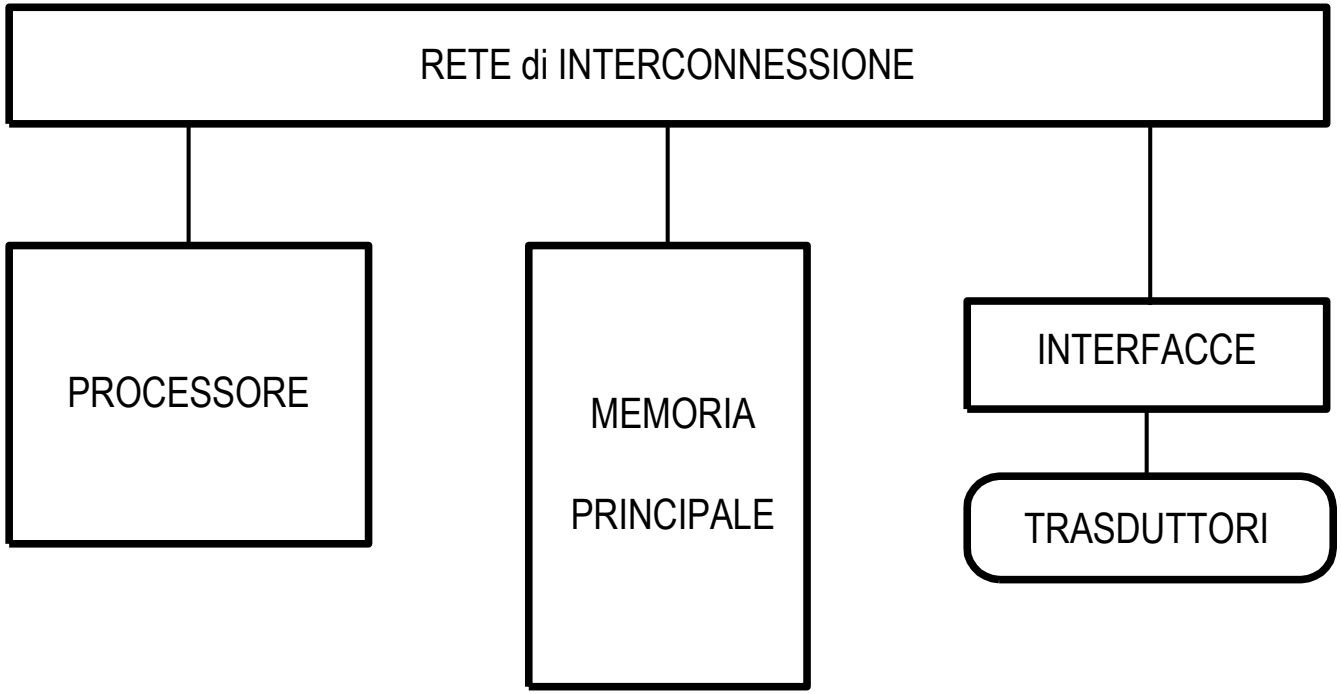
IEEE 754-2008 aggiunge:
rappresentazione in ***quadruple precision*** su **128 bit** con ***K* = 15** e ***G* = 112**

massimo modulo $\cong 2^{(bias+2)} = 2^{+16385} \cong 1.2 \cdot 10^{+4932}$
minimo modulo $= 2^{-bias} = 2^{-16383} \cong 1.6 \cdot 10^{-4932}$

Numeri Reali - Virgola mobile (10/10)



Architettura di von Neumann (1946)



Funzionamento

La memoria contiene dati e programmi (istruzioni) codificati in forma binaria

Il processore ripete all'infinito le azioni seguenti :

- preleva una nuova istruzione dalla memoria
- la decodifica
- la esegue

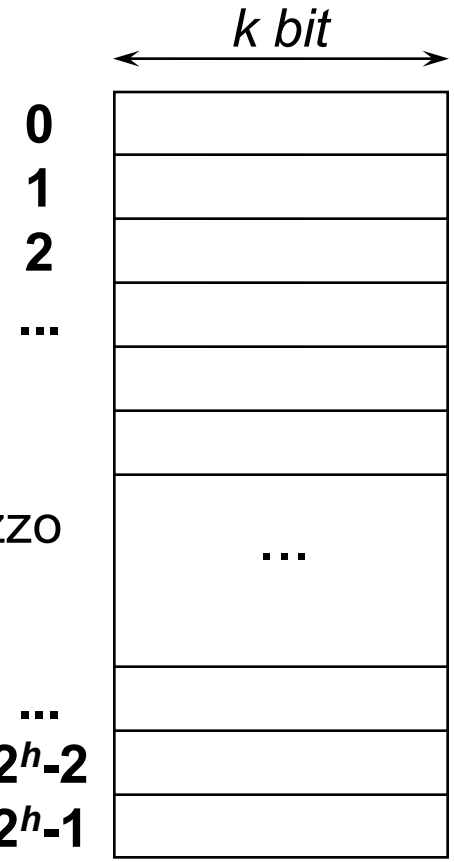
  L'esecuzione di un'istruzione può comportare

  » Elaborazione e/o Trasferimento (memoria $\leftrightarrow$ processore, I/O $\leftrightarrow$ processore)

Le periferiche permettono al calcolatore di interagire con il mondo esterno

Struttura logica della memoria

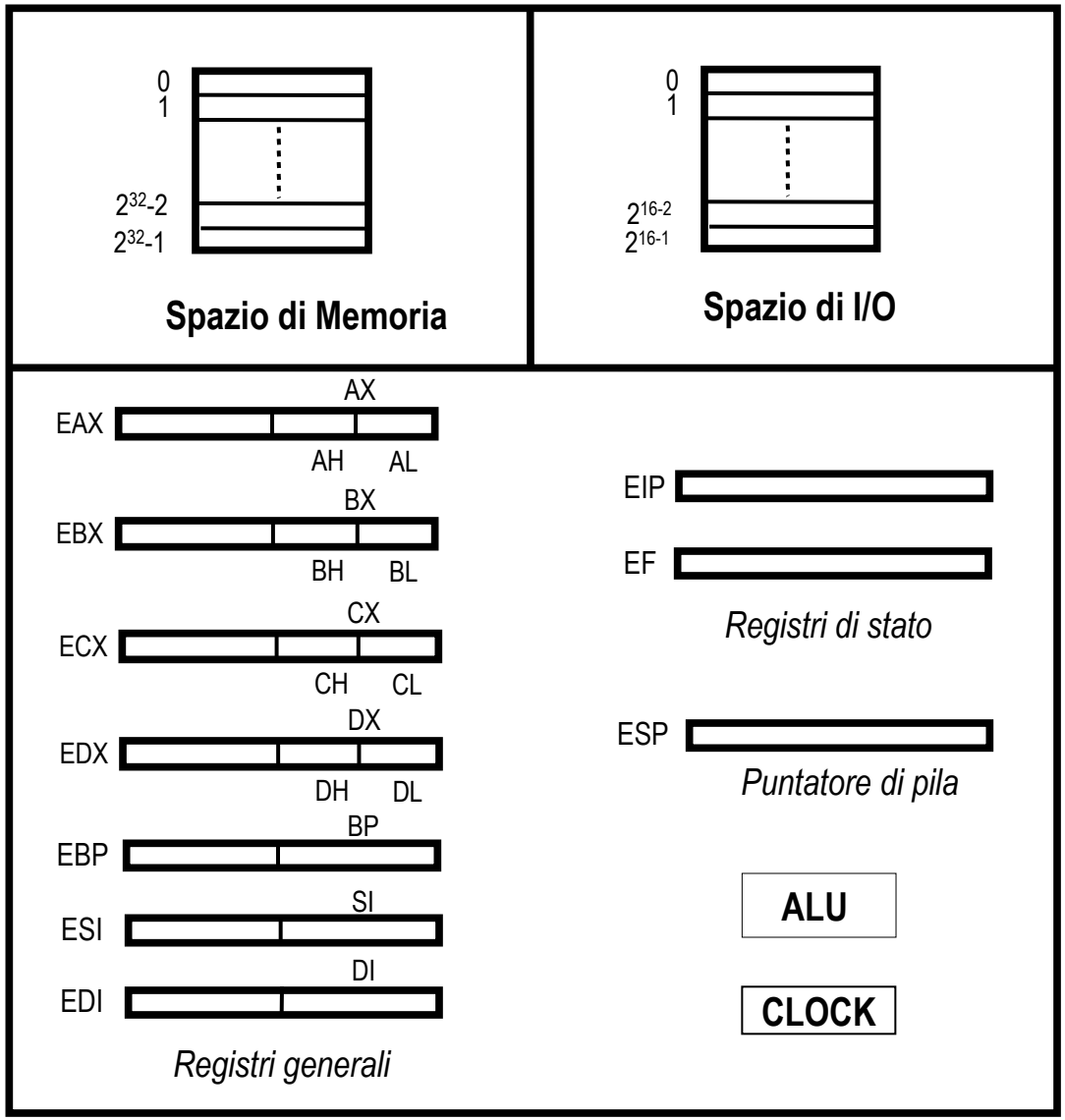
- 2^h *celle* di *k* bit ognuna
- Ogni cella ha un indirizzo



OPERAZIONI

1. LETTURA di UNA cella
2. SCRITTURA di UNA cella

Struttura logica del processore (1)



Livelli di astrazione dei Linguaggi

Esistono linguaggi a vari livelli di astrazione

Linguaggio macchina — sequenze di bit (difficile da leggere e capire)

```
0001001000110100
```

Linguaggio Assembler — istruzioni macchina espresse con nomi simbolici (dipendente dalla macchina)

```
MOV $35, %AL
```

Linguaggi ad Alto livello indipendenti dalla macchina

```
int main()
{
    …
}
```

Istruzioni MOV e JMP

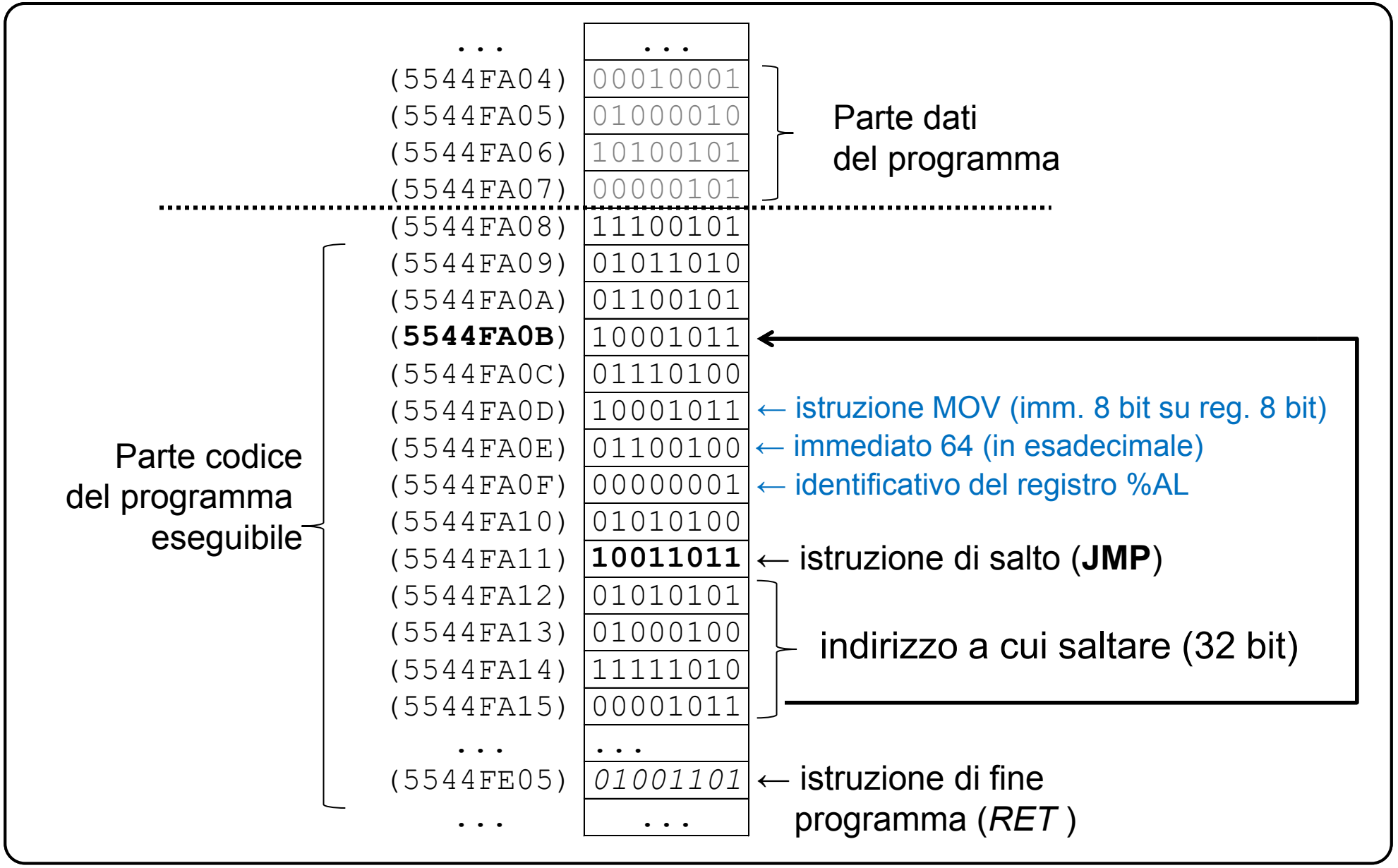
L'istruzione MOV del processore DP.86.32 permette di:

- Copiare un valore (immediato) in un registro
  Es: MOV $0x64, %AL
- Copiare il contenuto di una cella di memoria in un registro:
  Es: MOV (5544FA04), %BH
- Copiare il contenuto della cella puntata da un registro a 32 bit in un altro registro a 8 bit:
  Es: MOV (%EBX), %CH
- Copiare il contenuto delle **2 celle consecutive** di cui la prima è puntata da un registro a 32 bit in un altro registro a 16 bit:
  Es: MOV (%EDX), %DX
- Copiare il contenuto delle **4 celle consecutive** di cui la prima è puntata da un registro a 32 bit in un altro registro a **32 bit**:
  Es: MOV (%ECX), %EAX

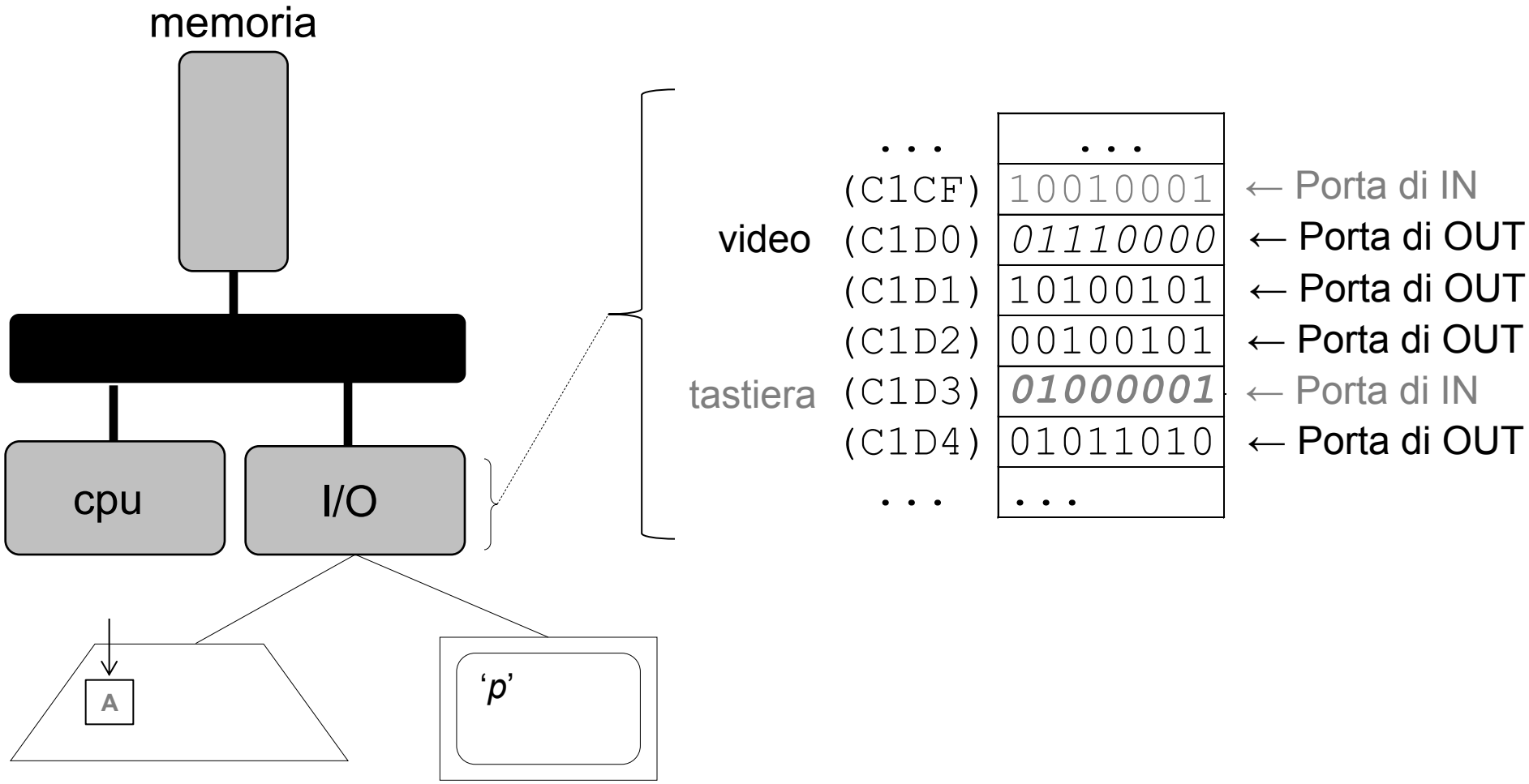
L'istruzione JMP del processore DP.86.32 permette di saltare all'indirizzo (sempre a 32 bit) specificato:

Es: JMP 5544FA0B

Esempio di programma in memoria



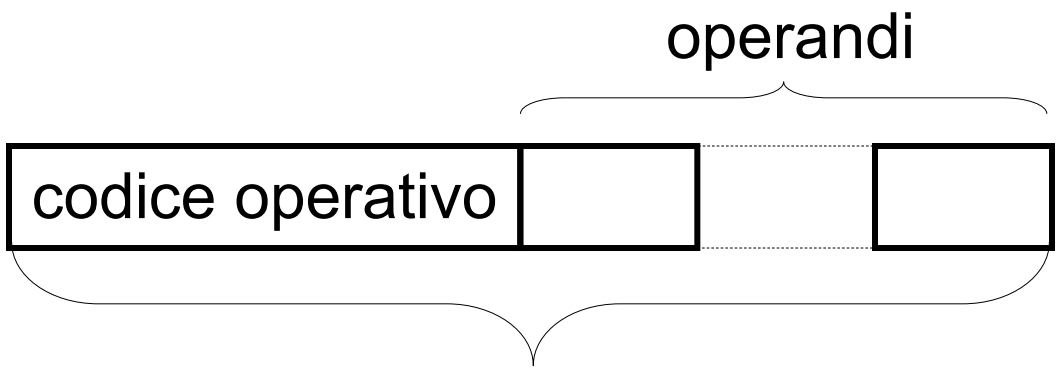
Lo spazio di ingresso/uscita




```
IN (C1D3), %AL          # carica in AL l'esad. 41, ossia 'A' (cfr cod. ASCII)
OUT $0x70, (C1D0)       # stampa a video 0x70, ossia il carattere 'p'
```

Formato delle istruzioni (1/2)

Le istruzioni sono codificate come sequenze di bit.



Esempio

Codice Operativo	**Operandi**	
`10001011`	`01100100`	`00000001`
MOV	immediato 0x64	%AL

Formato delle istruzioni (2/2)

OPCODEsize source destination

OPCODEsize source

OPCODEsize destination

Dove size può essere:

B (byte): 8 bit
W (word): 16 bit
L (Long): 32 bit

Esempi:

MOVB (%EBX), %AL
MOVW (%EBX), %AX
MOVL (%EBX), %EAX

Esempio di programma in lunguaggio assembler

```
3F0D0100    00000000                Dato da elaborare (una WORD)
3F0D0102    00000000                Risultato (un BYTE)
3F0D0103
...
...
40B1A200    MOV   $0, %AL           Azzera  il contatore AL
40B1A203    MOV   (3F0D0100), %DX   Copia in DX il dato da elaborare
40B1A207    CMP   %DX, $0           Confronta il contenuto di DX con 0
40B1A211    JE    40B1A232          Salta se uguale
40B1A215    SHL   %DX               Trasla a sinistra il contenuto di DX
40B1A217    JC    40B1A225          Salta se CF è settato all'indirizzo 225
40B1A221    JMP   40B1A207          Salta incondizionatamente a 207
40B1A225    ADD   $1, %AL           Incrementa il contatore AL
40B1A228    JMP   40B1A207          Salta incondizionatamente
40B1A232    MOV   %AL, (3F0D0102)   Memorizza il risultato
40B1A236    HLT                     ALT
…
```

Istruzioni operative (1)

Istruzioni per il trasferimento di dati

MOV	Movimento (ricopiamento)
IN	Ingresso dati
OUT	Uscita dati
PUSH	Immissione di una word nella pila
POP	Estrazione di una word dalla pila

Istruzioni aritmetiche

ADD	Somma (fra interi oppure naturali)
SUB	Sottrazione (fra interi oppure naturali)
CMP	Confronto (fra interi oppure naturali)
MUL	Moltiplicazione (fra naturali)
DIV	Divisione (fra naturali)
IMUL	Moltiplicazione (fra interi)
IDIV	Divisione (fra interi)

Istruzioni operative (2)

Istruzioni logiche

NOT	Not logico bit a bit
AND	And logico bit a bit
OR	Or logico bit a bit

Istruzioni di traslazione/rotazione

SHL	Traslazione a sinistra
SHR	Traslazione a destra
ROL	Rotazione a sinistra
ROR	Rotazione a destra

Istruzioni di controllo

Istruzioni di salto

JMP	Salto incondizionato
Jcond	Salto sotto condizione

Istruzioni per la gestione dei sottoprogrammi

CALL	Chiamata di sottoprogramma
RET	Ritorno da sottoprogramma

Istruzione di alt

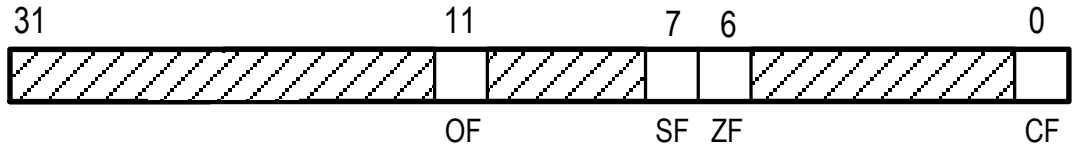
HLT	Alt

Controllo del flusso: salto condizionato e registro dei flag

Registri di stato:

Extended Instruction Pointer EIP: registro che contiene l'indirizzo della prossima istruzione da eseguire

Extended Flag register EF



Carry Flag CF, Zero Flag ZF

SignFlag SF, Overflow Flag OF

Esempio di test del contenuto dei flag:

Es1: CMP $0, %AL

JZ *indirizzo_di_memoria*

Es2: ADD %AL,%BL

JO *indirizzo_di_memoria*

NB:

Alcune istruzioni influenzano i flag, altre no

Ad esempio: la ADD influenza sia CF che OF, ma

- CF va testato nel caso si lavori sui naturali
- OF va testato nel caso si lavori sugli interi

Labels

Ogni istruzione assembler inizia ad un certo indirizzo in memoria (l'indirizzo della cella contenente il primo byte dell'istruzione)

L'indirizzo assoluto di una istruzione sarà noto solo a tempo di esecuzione, quando il programma assembler, reso eseguibile dall'assemblatore e dal linker, verrà caricato in memoria.

Il programmatore può creare delle etichette (dette *labels*), che possono essere utilizzate dal programmatore per indicare l'indirizzo di una istruzione. Sarà compito dell'assemblatore e del linker sostituire l'etichetta con l'indirizzo fisico assoluto.

Esempio di un programma che stampa 5 asterischi a video (il sottoprogramma *outchar* stampa a video il carattere la cui codifica ASCII si trova in %AL):

```
_main:    MOV   $5, %AH
          MOV   $'*', %AL
label:    CALL  outchar        # label conterrà l'indirizzo dell'istruzione
          DEC   %AH            # CALL outchar
          JNZ   label
          RET
.INCLUDE "utility"
```

Le direttive .BYTE, .FILL e .ASCII

In assembler si può allocare memoria «statica» per delle «variabili». Si tratta dell'equivalente C/C++ delle variabili globali.

Una «variabile» assembler è una etichetta (che corrisponde all'indirizzo della prima cella di memoria in cui inizia la «variabile») seguita da una direttiva assembler che riserva area di memoria.

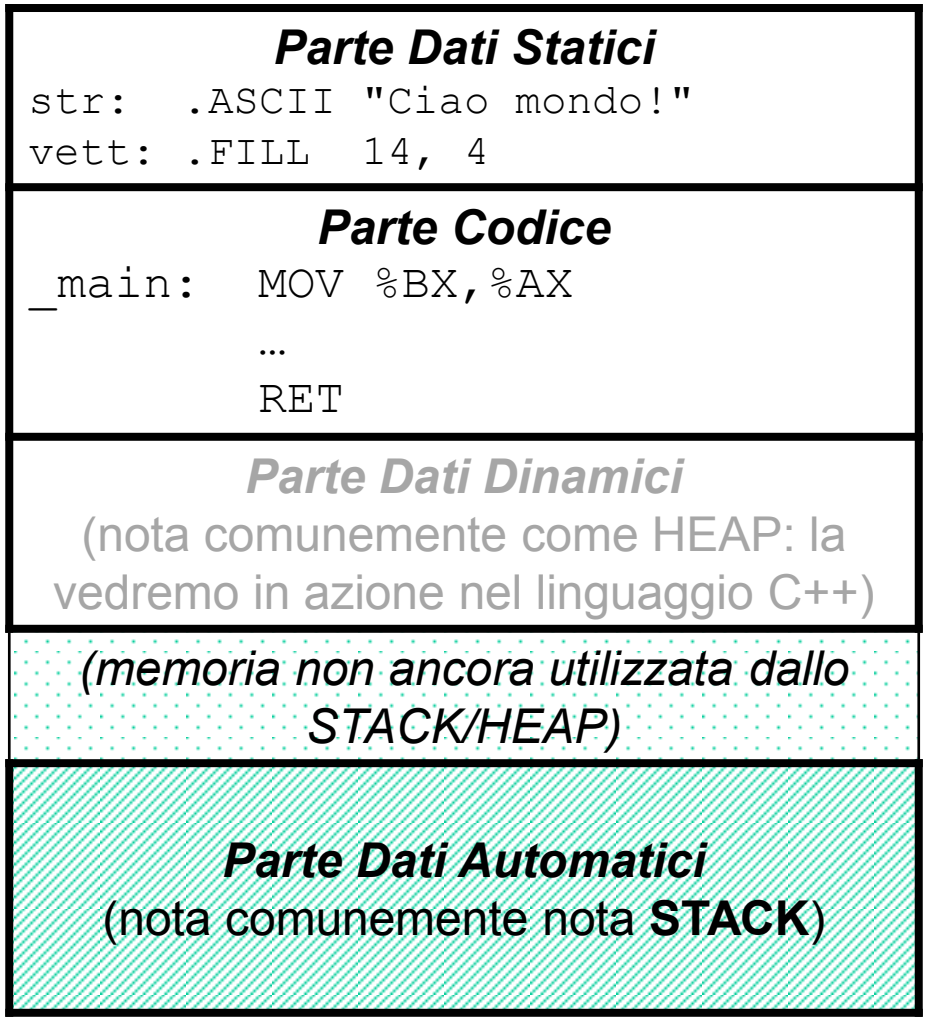
```
v1:    .BYTE 0x6F,'k' # alloca un vettore di 2 byte ad "ok"
v2:    .FILL   12, 1  # alloca un vettore di 12 byte (1=byte)
v3:    .FILL   25, 2  # alloca un vettore di 25 word (2=word = 2byte)
v4:    .FILL   10, 4  # alloca un vettore di 10 long (4=long = 4byte)
str:   .ASCII "Ciao Mondo"

_main:
       MOV   $str, %EBX  # copia l'indirizzo di str in EBX
       MOV   $10,  %CX
       CALL  outmess
       RET
```

NB: Le direttive .BYTE e .ASCII calcolano il numero di celle di memoria da allocare ed inoltre provvedono anche alla loro inizializzazione. La .FILL non inizializza.

Lo Stack (1/7)

- Lo STACK è un'area di memoria a disposizione dei programmi (sia Assembler che C/C++)
- Viene gestita mediante strategia LIFO (last in-first out): un nuovo dato può essere immesso oppure prelevato dalla cima allo STACK
- Sfrutta una registro speciale del processore, l'Extended Stack Pointer (ESP)
- L'ESP punta sempre alla prima locazione libera dello STACK

Struttura di un programma eseguibile in memoria



Le istruzioni PUSH e POP

- L'istruzione PUSH salva il contenuto di un registro a 32 bit nelle 4 celle in cima allo STACK e decrementa ESP di 4

  Esempio:

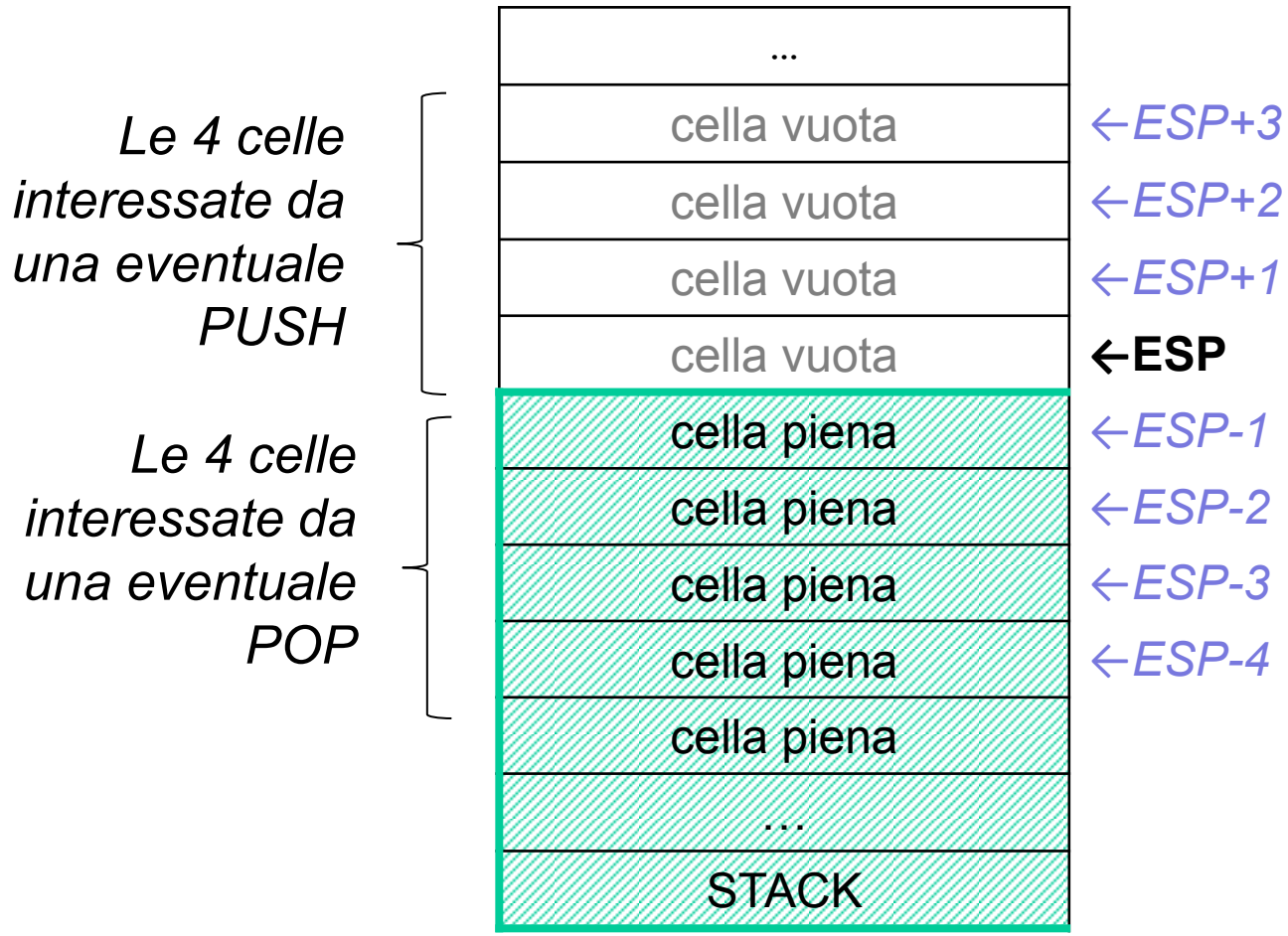
  ```
  PUSH %EAX
  ```

- L'istruzione POP recupera il contenuto delle 4 locazioni in cima allo STACK, lo salva in un registro a 32 bit e incrementa ESP di 4

  Esempio

  ```
  POP %EBX
  ```

Funzionamento delle istruzioni PUSH e POP



L'istruzione CALL

- L'istruzione CALL permette di andare ad eseguire un sottoprogramma
- Esempio di chiamata del sottoprogramma `inchar`:

```
04AAFFB3  CALL inchar
04AAFFB8  INC  %EAX
          ...
```

← *quando si arriva ad eseguire la CALL in EIP ci sarà indirizzo dell'istruzione INC (ad esempio, 04AAFFB8)*

- Concettualmente la CALL equivale alle seguenti istruzioni:

```
PUSH %EIP      # salva 04AAFFB8 sullo STACK
JMP 1E22AF0C   # se inchar inizia a 1E22AF0C
```

NB: la `PUSH %EIP` non si può fare esplicitamente in quanto non fa parte del set delle istruzioni del processore!

L'istruzione RET

- L'istruzione RET permette di tornare ad eseguire il sottoprogramma chiamante a partire dall'istruzione successiva alla CALL

Esempio:

```
1E22AF0C  inchar: IN (0AFB), %AL
                  ...
                  RET
```

- Concettualmente, effettua le seguenti operazioni:

```
POP %EIP
JMP %EIP
```

Lo Stack (7/7)

Sintetizzando, le istruzioni che utilizzano lo STACK sono:
CALL, PUSH, POP, e RET

NB:

- Tutte modificano l'ESP
- Alcune modificano anche l'EIP (CALL e RET)

Sottoprogrammi (1/2)

Qualunque blocco di istruzioni assembler consecutive che inizia con una label e termina con una RET.

Esempio di un sottoprogramma che stampa a video un asterisco:

```
subprog:  PUSHAD          # salva il contenuto di tutti i registri gen.
          MOV '*', %AL
          CALL outchar
          POPAD           # ripristina il contenuto di tutti i reg. gen.
          RET
```

Le informazioni fra il sottoprogramma chiamante (ad esempio il `_main`) e quello chiamato avviene tramite registri.

```
subprog:  CALL outchar # stampa il carattere messo in %AL dal chiamante
          CALL inchar  # legge un nuovo carattere e lo restituisce in %AL
          RET
```

Il sottoprogramma principale: _main

Ogni programma Assembler deve prevedere un sottoprogramma principale, denominato `_main`.

Quando il file ascii (con estensione .s) viene compilato per generare il file oggetto file.o e poi linkato in un .exe, la prima istruzione assembler che verrà eseguita sarà quella che si trova all'indirizzo dell'etichetta _main

Sottoprogrammi (3/3)

Alcuni sottoprogrammi di utilità, contenuti nel file assembler `"utility"`

Input ed output di caratteri e stringhe

`inchar` aspetta un carattere dalla tastiera e mette il suo codice ASCII in AL

`outchar` stampa a video il carattere contenuto in %AL

`outmess` stampa a video gli N caratteri che si trovano in memoria a partire dall'indirizzo contenuto in EBX. N è il naturale contenuto in CX.

Input ed output di naturali e interi su 8 bit

`inbyte` aspetta che vengano inseriti due caratteri (entrambi fra 0-9 o A-F) e poi, interpretandoli come cifre esadecimali, assegna gli 8 bit di %AL. Esempio 'E3' => in AL ci finisce 11100011

`outbyte` stampa a video i due caratteri ASCII corrispondenti alle due cifre esadecimali associate al corrente contenuto di %AL. Esempio: se il contenuto di AL è 01101010 a video viene stampata la stringa `"6A"`

`newline` Porta il cursore all'inizio della linea seguente (`CALL newline` equivale al `cout<<endl`)

Definizione di informatica

Informatica (definizione informale): è la scienza della rappresentazione e dell'elaborazione dell'informazione

Informatica (definizione formale dell'Association for Computing Machinery - ACM): è lo studio sistematico degli algoritmi che descrivono e trasformano l'informazione, la loro teoria e analisi, il loro progetto, e la loro efficienza, realizzazione e applicazione.

Algoritmo: sequenza precisa e finita di operazioni, comprensibili e perciò eseguibili da uno strumento informatico, che portano alla realizzazione di un compito.

Esempi di algoritmi:

- Istruzioni di montaggio di un elettrodomestico
- Somma in colonna di due numeri
- Bancomat

Calcolatore elettronico come risolutore di problemi

Compito dell'esperto informatico: data la descrizione di un problema, produrre algoritmi (cioè capire la sequenza di passi che portano alla soluzione del problema) e codificarli in programmi.

-La descrizione di un problema non fornisce in generale un modo per risolverlo.

- La descrizione del problema deve essere chiara e completa.

Calcolatori Elettronici come **esecutori di algoritmi**: gli algoritmi vengono descritti tramite programmi, cioè sequenze di istruzioni scritte in un opportuno linguaggio comprensibile al calcolatore.

Algoritmo (1)

Algoritmo: sequenza precisa (non ambigua) e finita di operazioni, che portano alla realizzazione di un compito.

Le operazioni utilizzate appartengono ad una delle seguenti categorie:

1. **Operazioni sequenziali**

   Realizzano una singola azione. Quando l'azione è terminata passano all'operazione successiva.

2. **Operazioni condizionali**

   Controllano una condizione. In base al valore della condizione, selezionano l'operazione successiva da eseguire.

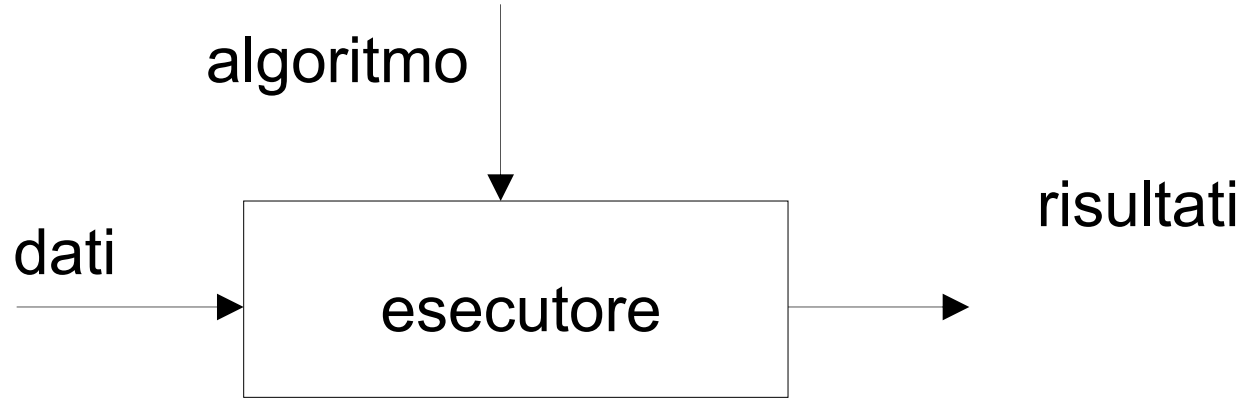
3. **Operazioni iterative**

   Ripetono l'esecuzione di un blocco di operazioni, finchè non è verificata una determinata condizione.

Algoritmo (2)

L'esecuzione delle azioni nell'ordine specificato dall'algoritmo consente di risolvere il problema.

Risolvere il problema significa produrre risultati a partire da dati in ingresso



L'algoritmo deve essere applicabile ad un qualsiasi insieme di dati in ingresso appartenenti al dominio di definizione dell'algoritmo (se l'algoritmo si applica ai numeri interi deve essere corretto sia per gli interi positivi che per gli interi negativi)

Calcolo equazione ax+b = 0

- leggi i valori di *a* e di *b*
- calcola –*b*
- dividi quello che hai ottenuto per *a* e chiama *x* il risultato
- stampa *x*

Calcolo del massimo fra due numeri

- leggi i valori di *a* e di *b*
- **se** *a* > *b* stampa *a* **altrimenti** stampa *b*

Algoritmo (4)

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*

Stabilire se una parola P precede alfabeticamente una parola Q.
Ipotesi: P e Q di uguale lunghezza > = 1

leggi *P* e *Q;* **inizializza** trovato a **0**

- **ripeti finché** (trovato vale 0 e lettere non finite)
  se prima lettera di *P* < prima lettera di *Q*
  allora scrivi vero; trovato = 1;
  altrimenti **se** prima lettera di *P* > prima lettera di *Q*
  allora scrivi falso; trovato = 1;
  altrimenti togli da *P* e da *Q* la prima lettera
- **se** trovato vale 0 **allora** scrivi falso

Algoritmo (5)

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 in ingresso, deve essere finito

Algoritmi equivalenti

- hanno lo stesso dominio di ingresso
- hanno lo stesso dominio di uscita
- in corrispondeza degli stessi valori del dominio di ingressso producono gli stessi valori del dominio di uscita

- Due algoritmi equivalenti
  - Forniscono lo stesso risultato, ma possono avere diversa efficienza e possono essere profondamente diversi

Esempio: calcolo del Massimo Comun Divisore (MCD) fra due interi M e N

Algoritmo 1

- Calcola l'insieme A dei divisori di M
- Calcola l'insieme B dei divisori di N
- Calcola l'insieme C dei divisori comuni
- il massimo comun divisore è il massimo divisore contenuto in C

Algoritmo 2 (di Euclide)

Se due numeri, m e n, sono divisibili per un terzo numero, x, allora anche la loro differenza è divisibile per x.

Per dimostrarla, si può utilizzare la proprietà distributiva.

Supponiamo m>n.

m=kx

n=hx

m-n=kx-hx=x(k-h)

Dunque si può dire che: **MCD(m,n) = MCD((m-n),n)**

Algoritmo

- **ripeti finché** (*M != N*):
  - **se** M > N, sostituisci a *M* il valore *M-N*
  - **altrimenti** sostituisci a *N* il valore *N-M*

- il massimo comun divisore corrisponde a M (o a N)

Algoritmo (7)

Proprietà essenziali degli algoritmi:

Correttezza:

- un algoritmo è corretto se esso perviene alla soluzione del compito cui è preposto, senza difettare in alcun passo fondamentale**.**

Efficienza:

- un algoritmo è efficiente se perviene alla soluzione del compito cui è preposto nel modo più veloce possibile, compatibilmente con la sua correttezza**.**

Programmazione

La formulazione testuale di un algoritmo in un linguaggio comprensibile ad un calcolatore è detta PROGRAMMA.

Ricapitolando, per risolvere 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 al calcolatore: LINGUAGGIO DI PROGRAMMAZIONE

Linguaggi di Programmazione (1)

Perché non usare direttamente il linguaggio naturale?

Il LINGUAGGIO NATURALE è un insieme di parole e di regole per combinare tali parole che sono usate e comprese da una comunità di persone

- non evita le ambiguità
- non si presta a descrivere processi computazionali automatizzabili

Occorre una nozione di linguaggio più precisa.

Un LINGUAGGIO di programmazione è una notazione formale che può essere usata per descrivere algoritmi.

Si può stabilire quali sono gli elementi linguistici primitivi, quali sono le frasi lecite e se una frase appartiene al linguaggio.

Linguaggi di Programmazione (2)

Un linguaggio è caratterizzato da:

SINTASSI - insieme di regole formali per la scrittura di programmi, che fissano le modalità per costruire frasi corrette nel linguaggio

SEMANTICA - insieme dei significati da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio

- a parole (poco precisa e ambigua)
- mediante azioni (semantica operazionale)
- mediante funzioni matematiche (semantica denotazionale)
- mediante formule logiche (semantica assiomatica)

Definizione di linguaggio

Alfabeto V (lessico)
- insieme dei simboli con cui si costruiscono le frasi

Universo linguistico V*
- insieme di tutte le frasi (sequenze finite) di elementi di V

Linguaggio L su alfabeto V
- un sottoinsieme di V*

Come definire il sottoinsieme di V* che definisce il linguaggio?

Specificando in modo preciso la sintassi delle frasi del linguaggio TRAMITE una **grammatica formale**

Grammatiche

Grammatica G = <V, VN, P, S>

V	insieme finito di simboli terminali
VN	insieme finito di simboli non terminali
P	insieme finito di regole di produzione
S	simbolo non terminale detto simbolo iniziale

Data una grammatica G, si dice Linguaggio LG generato da G l'insieme delle frasi di V

- Derivabili dal simbolo iniziale S
- Applicando le regole di produzione P

Le frasi di un linguaggio di programmazione vengono dette programmi di tale linguaggio.

Grammatica BNF (1)

GRAMMATICA BNF (Backus-Naur Form) è una grammatica le cui regole di produzione sono della forma

X ::= A

X simbolo non terminale

A sequenza di simboli (terminali e non terminali)

Una grammatica BNF definisce quindi un linguaggio sull'alfabeto terminale V mediante un meccanismo di derivazione (riscrittura)

A deriva da ***X*** se esiste una sequenza di derivazioni da ***X*** ad A

X ::= A_1
A_2 unica regola che indica l'**alternativa** fra A_1, …, A_n
…
A_n

Grammatica BNF (2)

G = <V, VN, P, S>

V = {lupo, canarino, bosco, cielo, mangia, vola, canta, ., il, lo}

VN = {***frase***, ***soggetto***, ***verbo***, ***articolo***, ***nome*** }

S = ***frase***

Produzioni P

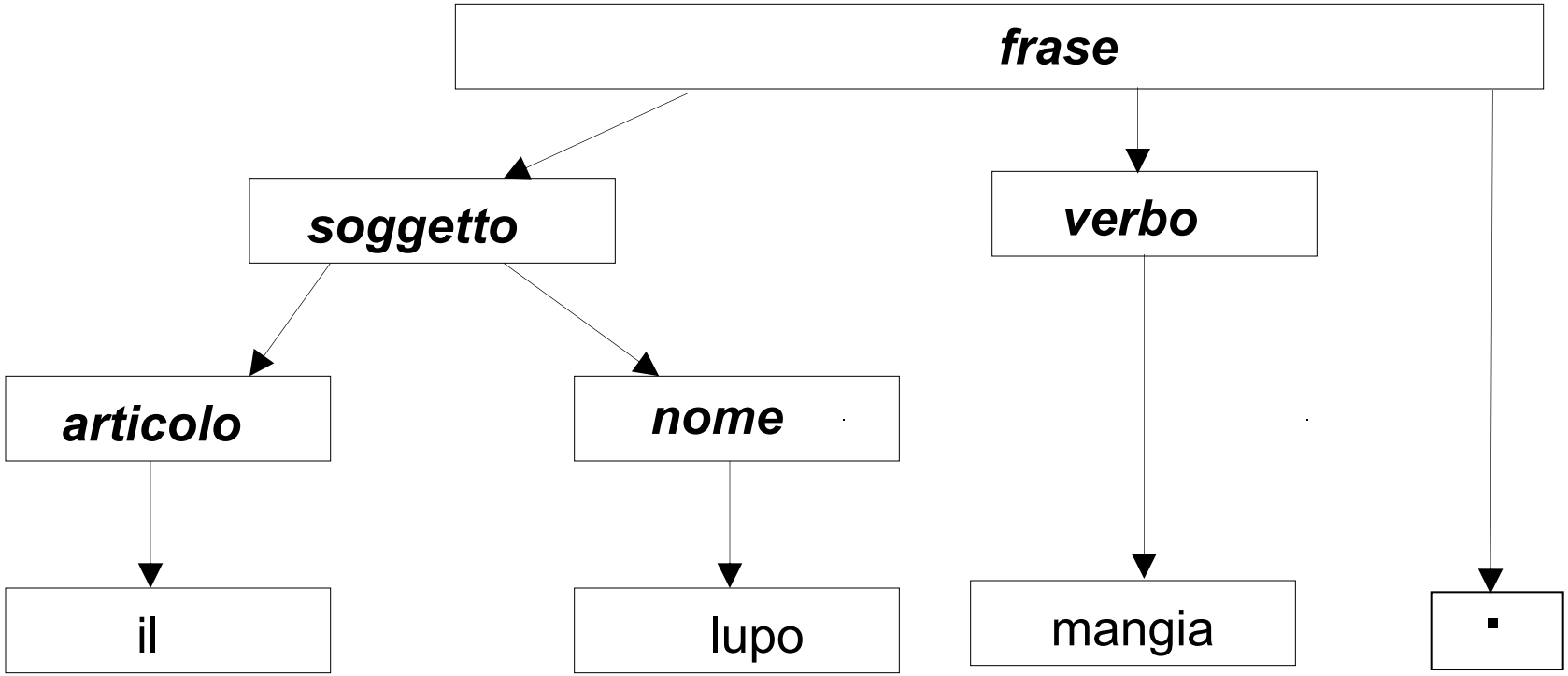
frase	::=	***soggetto verbo*** .
soggetto	::=	***articolo nome***
articolo	::=	il
		lo
nome	::=	lupo
		canarino
		bosco
		cielo
verbo	::=	mangia
		vola
		canta

Esempio: derivazione della frase
"il lupo mangia."

frase -> ***soggetto verbo***.
-> ***articolo nome verbo***.
-> il ***nome verbo***.
-> il lupo ***verbo***.
-> il lupo mangia.

derivazione left-most

Albero sintattico



Albero sintattico: albero che esprime il processo di derivazione di una frase usando una data grammatica

Esistono programmi per generare analizzatori sintattici per linguaggi descritti con BNF

Sintassi e semantica

Scrivere un programma sintatticamente corretto non implica che il programma faccia quello per cui è stato scritto

La frase “il lupo vola” è sintatticamente corretta ma non è semanticamente corretta (non è significativa)

```
// Somma di due numeri interi inseriti da tastiera

int main()
{
    int a, b;
    cout << “immettrere due numeri” << endl;
    cin >> a;  cin >> b;
    int c = a + a;  ←
    cout << “Somma: “ << c << endl;
    return 0;
}
```

Approccio compilato

Sviluppo di un programma (approccio compilato):

- editing: scrivere il testo e memorizzarlo su supporti di memoria permanenti
- compilazione
- linking
- esecuzione

Compilatore: traduce il programma sorgente in programma oggetto

- ANALISI programma sorgente
  - analisi lessicale
  - analisi sintattica
- TRADUZIONE
  - generazione del codice
  - ottimizzazione del codice

Esempi: C, C++, Pascal…

Approccio Interpretato

Sviluppo di un programma (approccio interpretato):

- editing: scrivere il testo e memorizzarlo su supporti di memoria permanenti
- interpretazione
  - ANALISI programma sorgente
    - analisi lessicale
    - analisi sintattica
  - ESECUZIONE

ESEMPIO: Basic, Java