

# Rappresentazione dei dati

Andrea Passerini  
passerini@dsi.unifi.it

Conoscenze informatiche e relazionali  
Corso di laurea in Scienze dell'Ingegneria Edile

# Codica analogica contro codifica digitale

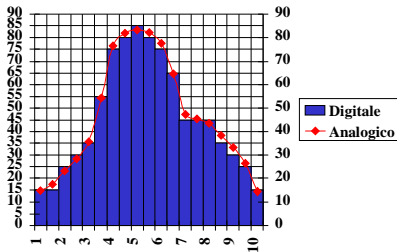
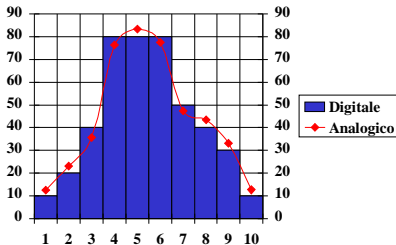
- Come è meglio codificare l'informazione per la sua elaborazione automatica mediante un calcolatore ?
- Grandezze naturali come l'intensità di un suono assumono valori *continui*.
  - Una codifica si dice *analogica* se mantiene un'analogia tra la struttura dell'entità di informazione e struttura della configurazione (possiamo parlare di codifica *continua*).
  - Una codifica si dice *digitale* se impone un numero di configurazioni distinte ammissibili e converte un'entità di informazione in una di queste configurazioni mediante una regola di codifica (possiamo parlare di codifica *discreta*).

# Codica analogica contro codifica digitale

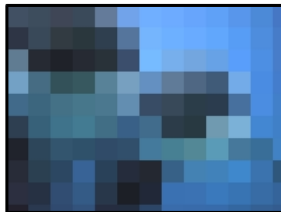
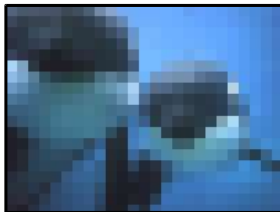
- Qualsiasi sistema fisico (incluso il calcolatore) è sottoposto all'influenza dell'ambiente circostante che ne perturba la configurazione introducendo *rumore*.
  - E' necessario rendere il sistema fisico che elabora l'informazione il più immune possibile al rumore.
  - La codifica analogica è fortemente sensibile al rumore, poichè tutte le configurazioni sono lecite e non si può distinguere la componente di informazione dal contributo dovuto al rumore.
  - Minore è il numero di configurazioni possibili maggiore è la possibilità di isolare l'informazione dal rumore.
  - E' questa la ragione principale del successo della codifica *binaria* nei calcolatori elettronici.

# Campionamento

Permette di approssimare in maniera arbitrariamente accurata informazione continua (analogica).



# Esempio



# Unità di misura per l'informatica

- Un *bit* (b) rappresenta una cifra binaria. E' l'unità minima di informazione.
- Un *Byte* (B) è costituito da 8 bit. Permette di codificare 256 entità di informazione distinte (e.g. caratteri dell'alfabeto, segni di interpunzione)
- Una *parola* (*word*) rappresenta l'insieme di Byte che possono essere trattati da un elaboratore in un'operazione. Il numero di Byte di una parola dipende dall'elaboratore, ad esempio 2, 3, 4, 8 corrispondenti a 16, 24, 32 o 64 bit.

## Multipli del Byte

- Kilobyte (KB) =  $2^{10}$  = 1024 Byte (e.g. un file di testo da 300KB)
- Megabyte (MB) = 1024KB  $\approx$  1 milione di Byte (e.g. un'immagine di 30MB)
- Gigabyte (GB) = 1024MB  $\approx$  1 miliardo di Byte (e.g. un hard disk da 80GB)
- Terabyte (TB) = 1024GB  $\approx$  1000 miliardi di Byte (e.g. un archivio da 20TB)

# Bit rate

- Esprime la velocità di trasferimento dati in bit per secondo (bps).
- Multipli
  - Kilobit per secondo (Kbps) = 1000bps (e.g. un modem a 56Kbps)
  - Megabit per secondo (Mbps) = 1000Kbps (e.g. una ADSL a 4Mbps)

## Nota sul bit rate

- La dimensione dei file viene generalmente espressa in multipli **binari** del **Byte** (e.g. 1MB =  $2^{10}$  KByte =  $2^{20}$  Byte = 1,048,576 Byte)
- Il bit rate viene generalmente espresso in multipli **decimali** del **bit** (e.g. 1Mbps = 1,000,000 bps)
- Quindi per scaricare un file da 4MB con un'ADSL a 4Mbps sono necessari (nella situazione ottimale):

$$\frac{4 \times 2^{20} \times 8}{4 \times 1000000} = 8.388608$$

ossia circa 8 secondi e mezzo.

- Questo non considerando la compressione e il tempo necessario a trasmettere l'intestazione dei pacchetti

# Hertz (Hz)

- Grandezza per misurare la rapidità (frequenza) dei dispositivi digitali
- Il nome deriva dal fisico Heinrich Rudolf Hertz
- 1 Hz corrisponde ad un ciclo o una oscillazione al secondo
- Multipli
  - Kilohertz (KHz) = 1000 Hz (si usa per misurare la frequenza di refresh dello schermo)
  - Megahertz (MHz) = 1000 KHz
  - Gigahertz (GHz) = 1000 MHz (si usa per misurare la frequenza di clock dei calcolatori)

# Hertz e clock

- Il clock è un dispositivo che funziona come un metronomo sincronizzando tutte le operazioni dei dispositivi digitali
- Nota:
  - L'unità di misura Hz si può usare come indicatore relativo della velocità di elaborazione di un computer
  - ovvero si può usare per paragonare due processori con la stessa architettura (e.g. un 486 a 100MHz contro un 486 a 120MHz)
  - NON si possono però fare paragoni fra processori diversi (e.g. RISC Motorola contro Pentium)

## Dots Per Inch (DPI) ossia punti per pollice

- Grandezza per misurare la densità di punti o *definizione* o *risoluzione*.
- Un pollice quadrato corrisponde ad un'area di 2.54 cm<sup>2</sup>
- Stampanti e schermi usano matrici di punti (*pixel*) per rappresentare immagini 2D.
- Maggiore è il numero di punti nell'unità di area maggiore è l'accuratezza con la quale si definisce un'immagine o un testo
- Esempio:
  - 300 dpi per le stampanti laser
  - 1200 dpi per gli scanner
- La risoluzione di uno schermo si indica riportando il numero di pixel visualizzati sul lato orizzontale e su quello verticale (e.g. 1280x800 o 860x640)

# Codifica dei dati

## Codifica binaria

- Qualsiasi informazione deve essere codificata in binario per poter essere trattata da un calcolatore.
- Vedremo come vengono codificati:
  - Numeri interi
  - Numeri “reali” (*real*)
  - Caratteri

# Codifica dei numeri

## Numero di bit fissato

- Il calcolatore può fare operazioni su due numeri solo se sono codificati con lo stesso numero di bit.
- Si fissa il numero di bit  $k$  con cui si rappresenta un certo insieme di numeri (e.g. interi a 8 bit)
- Ogni numero di tale insieme deve essere rappresentato con  $k$  bit, eventualmente aggiungendo zeri a sinistra
- Esempio per interi positivi a  $k = 8$  bit

$$(54)_{10} = (00110110)_2$$

# Rappresentazione di interi

## Codifica valore assoluto con segno

- Si riserva il bit più significativo (quello più a sinistra) al segno (0=positivo, 1= negativo).
- Si codifica con i restanti bit il modulo del numero.
- Esempio con  $k = 8$  bit:

$$(+54)_{10} = (00110110)_2$$

$$(-54)_{10} = (10110110)_2$$

- Problema: il *bit di segno* deve essere trattato in maniera diversa rispetto agli altri bit (complica i circuiti per somma e sottrazione).

# Rappresentazione di interi

## Codifica nella forma complemento a 2

- Si rappresenta ogni numero negativo tramite il complemento a 2 del corrispondente numero positivo.
- Esempi con  $k = 8$  bit:

$$\begin{aligned} (+33)_{10} &= (00100001)_2 & (+127)_{10} &= (01111111)_2 \\ (-33)_{10} &= (11011111)_2 & (-128)_{10} &= (10000000)_2 \end{aligned}$$

- E' come associare un peso  $-2^{k-1}$  invece di  $2^{k-1}$  al MSB (*most significant bit*)
- Per esempio:

$$(10110110)_2 = -128 + 32 + 16 + 4 + 2 = -74$$

- Il MSB sarà sempre 0 per numeri positivi e 1 per numeri negativi (*bit di segno*)

## Addizione nella forma complemento a 2

- Si tratta il bit di segno esattamente come gli altri bit:

$$\begin{array}{r} 25 \quad + \\ 31 \quad = \\ \hline 56 \end{array} \qquad \begin{array}{r} 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ + \\ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ = \\ \hline 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \end{array}$$

$$\begin{array}{r} -25 \quad + \\ -31 \quad = \\ \hline -56 \end{array} \qquad \begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ + \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ = \\ \hline (1) \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \end{array}$$

- Il riporto (*carry*) che si ottiene nel sommare numeri negativi viene ignorato.

## Addizione nella forma complemento a 2

Perché funziona con i negativi ?

- Il risultato della somma di due numeri negativi sarà ancora negativo  $\Rightarrow$  andrà ancora rappresentato in complemento a 2.
- Rappresenteremo quindi  $-X - Y$  come  $C_2(X + Y)$ .
- Vale la seguente relazione (numeri a  $k$  bit):

$$\begin{aligned}C_2(X + Y) &= 2^k - X - Y \\ &= (2^k - X) - Y = C_2(X) - Y \\ &= C_2(X) - (2^k - C_2(Y)) \\ &= C_2(X) + C_2(Y) - 2^k\end{aligned}$$

- Sottrarre  $2^k$  dalla somma di  $C_2(X)$  e  $C_2(Y)$  equivale ad ignorare il bit di carry.

# Overflow

- Si verifica quando la somma di due numeri è al di fuori del rango (*range*) di valori permessi nella rappresentazione scelta per i numeri.
- Ad esempio il rango dei numeri interi rappresentati in complemento a 2 con  $k$  bit è  $[-2^{k-1}, 2^{k-1} - 1]$
- Il risultato dell'operazione non è corretto nel caso si verifichi un overflow.
- Gli elaboratori elettronici verificano la condizione di overflow quando effettuano l'addizione binaria e la segnalano mettendo ad 1 uno speciale *bit di overflow*.

## Esempio di overflow ( $k = 8$ )

- 71 e 60 sono entrambi rappresentabili in complemento a due su 8 bit (il rango è  $[-128, 127]$ ).
- La loro somma  $71+60=131$  non è rappresentabile
- Calcolando la somma binaria otteniamo:

$$\begin{array}{r} 71 \quad + \\ 60 \quad = \\ \hline 131 \end{array} \qquad \begin{array}{r} 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ + \\ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ = \\ \hline 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \end{array}$$

ossia  $-128+2+1=-125$  nella forma complemento a 2.

## Sottrazione nella forma complemento a 2

- Si calcola il complemento a 2 del sottraendo e si somma il risultato al minuendo.
- Ad esempio  $29 - 7 = 29 + (-7) = 22$ , ossia in binario:

<i>rappresenta 7</i>	0	0	0	0	0	1	1	1	
<i>complementa a 2</i>	1	1	1	1	1	0	0	1	+
<i>rappresenta 29</i>	0	0	0	1	1	1	0	1	=
<i>somma -7 a 29</i>	(1)	0	0	0	1	0	1	1	0

dove il bit di carry viene ignorato.

- Perché funziona ?

$$\text{Caso } X > Y \quad X - Y = X - (2^k - C_2(Y)) = X + C_2(Y) - 2^k$$

$$\text{Caso } X < Y \quad C_2(Y - X) = 2^k - (Y - X) = (2^k - Y) + X = C_2(Y) + X$$

## Traslazione logica (*shift logico*)

- Consiste nello spostare a destra (*shift logico a destra*) o a sinistra (*shift logico a sinistra*) i bit di un numero binario.
- Nella traslazione a destra, il bit meno significativo viene perso, quello più significativo viene posto a 0.
- Un numero binario traslato a destra viene diviso per 2.
- Nella traslazione a sinistra, il bit più significativo viene perso, quello meno significativo viene posto a 0.
- Un numero binario traslato a sinistra viene moltiplicato per 2.

## Operazioni eseguite in termini di altre operazioni

- Con la codifica in complemento a 2, la sottrazione si realizza con una complementazione (operazione semplicissima) ed un'addizione.
- Si può realizzare un unico circuito che effettui somme e sottrazioni con notevoli risparmi di costi.
- Una moltiplicazione può essere realizzata tramite una sequenza di addizioni e di traslazioni a sinistra.
- Una divisione può essere realizzata tramite una sequenza di sottrazioni e di traslazioni a destra.
- Le operazioni più semplici vengono eseguite da appositi circuiti (a livello *hardware*).
- Operazioni più complesse sono eseguite in termini di esecuzione di altre operazioni più semplici sotto il controllo di programmi (a livello *software*).

# Rappresentazione di numeri “reali” (*real*)

## Rappresentazione in virgola fissa

- Si stabilisce un numero di bit  $k_1$  da assegnare alla parte intera ed un numero di bit  $k_2$  da assegnare alla parte frazionaria.
- Ad esempio per numeri a 32 bit se ne assegnano 16 alla parte intera e 16 alla parte frazionaria.
- Adatta solo a casi particolari in cui l'intervallo di valori da rappresentare è noto a priori.
- Inadatta nella maggior parte delle applicazioni scientifiche.

## Rappresentazione in virgola mobile (*floating point*)

- Si parte dalla rappresentazione scientifica in cui il numero è il prodotto di due parti: una parte frazionaria ed un fattore di scala, che è una potenza del 10.
- Ad esempio il numero 23.5 può essere rappresentato come:

$$23.5 \times 10^0$$

$$2.35 \times 10^1$$

$$235 \times 10^{-1}$$

$$0.235 \times 10^2$$

...

...

- Rappresentazione scientifica normalizzata: la parte frazionaria ha la cifra più a sinistra diversa da 0 e subito seguita dal punto decimale (e.g.  $2.35 \times 10^1$ ).
- Nella rappresentazione floating point si rappresenta il numero come una coppia: la *mantissa* corrispondente alla parte frazionaria, la *caratteristica* (o *esponente*) che corrisponde all'esponente del fattore di scala.

# Rappresentazione floating point binaria

- Nel caso binario mantissa ed esponente sono rappresentati in binario, ed il fattore di scala è una potenza del 2.
- Esempi:

Numero	Rapp. norm.	Mantissa	Esponente
+101010.0	$+1.010100 \times 10^{101}$	+1.010100	+101
+0.000110	$+1.100000 \times 10^{-100}$	+1.100000	-100
-110.1100	$-1.101100 \times 10^{10}$	-1.101100	+010

## Codifica floating point binaria

- La mantissa è codificata in forma valore assoluto con segno.
- Il segno (della mantissa) viene riportato come primo bit dell'intera rappresentazione.
- Il primo bit della mantissa ed il punto di separazione non vengono rappresentati in quanto sono sempre gli stessi ("1.").
- La caratteristica è codificata nella *forma in eccesso a t*: con un numero binario senza segno ottenuto sommando all'esponente vero e proprio la costante  $t = 2^{k-1} - 1$  con  $k$  numero di bit riservati alla caratteristica.

## Esempio a 32 bit

*bit di segno*  $\rightarrow$  1  $\underbrace{01100000}_{\text{caratteristica}}$   $\underbrace{001110000000000000000000}_{\text{mantissa}}$

- La mantissa vale  $(-1.00111)_2 = -1.21875$ .
- Il numero binario scritto nella caratteristica è  $(01100000)_2 = 96$
- La caratteristica è codificata in  $k = 8$  bit. Dunque  $t = 2^{k-1} - 1 = 127$ .
- Il valore dell'esponente è  $96 - 127 = -31$ .
- Il numero rappresentato è quindi  $-1.21875 \times 2^{-31} \approx -5.675247 \times 10^{-10}$

## Codifica di caratteri

- Repertorio: insieme di caratteri considerati, definito mediante i nomi dei caratteri e magari una loro rappresentazione visiva.
- Numero di codice: tabella che associa ad ogni carattere un numero da un dato insieme di numeri naturali.
- Codifica: un metodo per associare a ciascun numero di codice una sequenza di bit.
- Nel caso più semplice ogni carattere ha un numero tra 0 e 127 e la codifica è semplicemente la codifica binaria del numero in 7 bit.

# Codice ASCII

- Acronimo di *American Standard Code for Information Interchange*.
- 7 bit per carattere, si possono rappresentare  $2^7 = 128$  caratteri distinti. I codici sono tipicamente scritti in notazione esadecimale.
- I codici da 0 a 1F sono usati per *caratteri di controllo*
- I codici da 20 a 7E sono usati per *caratteri stampabili*
- Ordine alfabetico: cifre 0-9 prima dei caratteri alfabetici, maiuscole prima delle minuscole (si rifletterà nell'ordinamento lessicografico delle stringhe).

# Tabella dei codici ASCII

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

## Limitazioni del codice ASCII

- I caratteri internazionali di numerose lingue europee (quali ad esempio è, ù, ç, å, æ, ü, ø) non sono rappresentabili
- nessuno dell'elevatissimo numero di simboli delle lingue asiatiche è rappresentabile
- La standardizzazione è importante: nella trasmissione e memorizzazione elettronica i caratteri sono rappresentati da insiemi di Byte ed è importante che il "trasmettitore" ed il "ricevente" adottino le stesse convenzioni!
- In assenza di opportune convenzioni, testi generati su un dato sistema possono risultare corrotti se visualizzati in un sistema diverso (capita facilmente con la posta elettronica).

# ISO Latin-1 (ISO 8859-1)

- Il repertorio contiene il repertorio ASCII come sottoinsieme e i codici per questi caratteri sono identici a quelli ASCII
- Il codice usa 8 bit: 256 caratteri distinti
- Contiene vari simboli usati dai linguaggi dell'Europa occidentale (Italiano, Francese, Spagnolo, Tedesco, Danese, etc.):

¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸  
¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð  
Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è  
é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

## La famiglia ISO 8859

- La famiglia contiene ben 15 alfabeti standard, tra cui ad esempio:

8859-2 (Latin-2) lingue dell'Europa centrale e orientale

8859-5 (Latin/Cyrillic) lingue Slave

8859-7 (Latin/Greek) Greco moderno

8859-9 (Latin-5) Turco

8859-10 (Latin-6) lingue nordiche (Islandese)

8859-15 (Latin-9) Latin-1 con l'Euro

# ISO 10646 e UNICODE



- ISO 10646 è uno standard internazionale che definisce lo UCS (Universal Character Set)
- Il repertorio è molto ampio e contiene decine di migliaia di caratteri già definiti, con spazio per espansioni future
- Estende ISO Latin-1 nello stesso senso in cui ISO Latin-1 estende ASCII
- UNICODE è uno standard definito dal consorzio UNICODE (fondato nel '92) e definisce un repertorio e una codifica pienamente compatibili con ISO 10646
- UNICODE implementa solo una parte di ISO 10646

# UNICODE

- Il vantaggio di UNICODE è di definire un insieme di caratteri adatti a trattare tutti i linguaggi, mentre la famiglia ISO 8859 definisce sottoinsiemi di caratteri adatti a trattare solo alcuni linguaggi alla volta
- Mentre con altri standard lo stesso carattere può avere codifiche diverse (a seconda dell'alfabeto usato), in UNICODE ogni carattere ha una codifica unica
- Ha una grande diffusione industriale (Apple, HP, IBM, Microsoft, Oracle, Sun, etc.)
- E' supportato da vari sistemi operativi ed internet browser più recenti.

# UNICODE

- Il codice usa 16 bit (fino a 64K caratteri)
- I primi 256 codici coincidono con quelli di ISO Latin-1
- Esempio di codifica per caratteri armeni:

	053	054	055	056	057	058
0		Հ 0540	Ր 0550		Տ 0570	Ր 0580
1	Ա 0531	Զ 0541	Յ 0551	Մ 0561	Ճ 0571	Գ 0581
2	Բ 0532	Ղ 0542	Ի 0552	Բ 0562	Պ 0572	Լ 0582
3	Գ 0533	Ճ 0543	Փ 0553	Գ 0563	Ճ 0573	Փ 0583
4	Դ 0534	Մ 0544	Ք 0554	Դ 0564	Մ 0574	Ք 0584