Rappresentazione dell’Informazione

Corrado Santoro

Dipartimento di Matematica e Informatica
santoro@dmi.unict.it

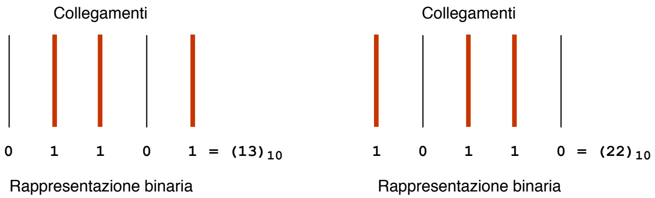
Corso di Architettura degli Elaboratori

Riassumiamo...

- Dal punto di vista elettrico, i circuiti di un calcolatore funzionano secondo il modello **OFF/ON**
- Per nostra comodità associamo i concetti OFF/ON ai simboli 0 e 1:
  - **OFF** = 0
  - **ON** = 1
- Possiamo dunque pensare ad un **insieme di stati OFF/ON** come alle cifre 0 e 1 di un **numero binario**

Circuiti e Numeri binari

- Consideriamo un insieme di k collegamenti elettrici, ognuno dei quali opera secondo il modello **OFF/ON**
- Rappresentiamo lo stato (elettrico) di ogni collegamento con una cifra binaria
- I k collegamenti possono dunque essere utilizzati per **rappresentare un qualunque intero** nell'intervallo $[0, 2^k - 1]$



BIT

- Una *cifra binaria* è detta **BIT** = **BInary digiT**
- Un singolo collegamento elettrico è in grado di rappresentare **1 BIT**

BYTE

- Un *combinazione di* **8 bit** è detta **BYTE**
- 8 collegamenti elettrici rappresentano **1 BYTE**
- **Un BYTE** è dunque in grado di rappresentare tutti gli interi nell'intervallo $[0, 2^8 - 1] = [0, 255]$

Definizioni e Terminologie

WORD

- Un *combinazione di* **16 bit** è detta **WORD** = 2 BYTE
- 16 collegamenti elettrici rappresentano **1 WORD**
- **Una WORD** è dunque in grado di rappresentare tutti gli interi nell'intervallo $[0, 2^{16} - 1] = [0, 65535]$

DOUBLE WORD

- Un *combinazione di* **32 bit** è detta **DOUBLE WORD** = 4 BYTE
- 32 collegamenti elettrici rappresentano **1 DOUBLE WORD**
- **Una DOUBLE WORD** è dunque in grado di rappresentare tutti gli interi nell'intervallo $[0, 2^{32} - 1] = [0, 4294967295]$

Osservazioni

- L'**ampiezza** dell'intervallo che intendiamo rappresentare è funzione del **numero di bit** che utilizziamo
- Ogni bit è poi "realizzato fisicamente" tramite un collegamento elettrico
- Poichè non è possibile realizzare fisicamente **infiniti collegamenti, non è possibile** rappresentare, in un calcolatore, *tutto l'insieme dei numeri naturali*
- La rappresentazione è sempre legata ad un **intervallo ben preciso** dipendente dal numero di bit utilizzati

Verso l'aritmetica modulare

- La rappresentazione è sempre legata ad un **intervallo ben preciso** dipendente dal numero di bit utilizzati
- Ma cosa accade se, applicando un'operazione ai nostri numeri, **superassimo** l'intervallo ammissibile?
- Ad esempio, cosa accade se, con 8 bit, riuscissimo ad effettuare (elettricamente) la **somma** $255 + 1$? Quale sarebbe il nostro risultato?

Esempio

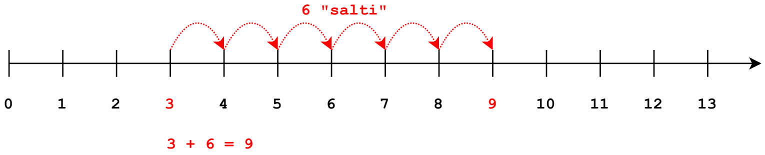
Riporti	1	1	1	1	1	1	1	1		
Primo addendo		1	1	1	1	1	1	1	1	+
Secondo addendo		0	0	0	0	0	0	0	1	=
Risultato (artimetico)	1	0	0	0	0	0	0	0	0	
Risultato reale		0	0	0	0	0	0	0	0	

C'è spazio solo per 8 bit, i bit in eccesso vengono **troncati**, dunque: $255 + 1 = 0$

Richiami di Aritmetica Intera

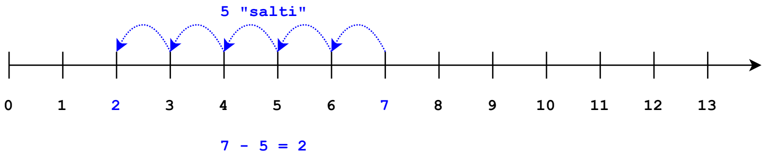
Linea dei numeri e l'operazione di somma

- Ci hanno insegnato che i numeri possono essere rappresentati lungo una **(semi-)retta**
- L'operazione di **somma (intera)** possiamo "visualizzarla" con il concetto di **salto al successivo**
- Partiamo dal *primo addendo* (sulla linea dei numeri) e facciamo un numero di **"salti al successivo"** pari al valore del *secondo addendo*
- Il numero a cui arriveremo sarà il **risultato della somma**
- Se consideriamo gli **interi non negativi**, la linea dei numeri sarà una semiretta che conterrà **infiniti numeri**



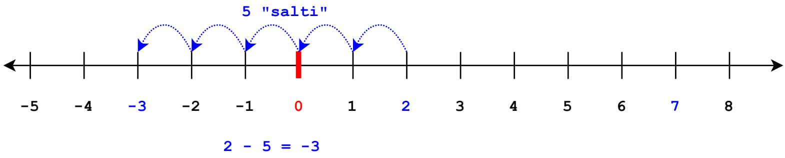
Linea dei numeri e l'operazione di sottrazione

- Analogamentem l'operazione di **sottrazione (intera)** possiamo "visualizzarla" con il concetto di **salto al precedente**
- Partiamo dal *minuendo* (sulla linea dei numeri) e facciamo un numero di **"salti al precedente** pari al valore del *sottraendo*
- Il numero a cui arriveremo sarà il **risultato della sottrazione**



Linea dei numeri e l'operazione di sottrazione

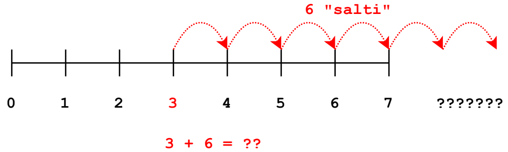
- Tuttavia cosa accade qualora, "saltando all'indietro", dovessimo superare lo 0?
- Estendiamo, a sinistra, la linea dei numeri introducendo i **numeri negativi**
- La linea dei numeri diventa dunque una **retta** che si estende da $-\infty$ a $+\infty$



Elementi di Aritmetica Modulare

Il "segmento" dei numeri

- L'insieme $\mathcal{Z}$ dei numeri relativi ha, per definizione, infiniti elementi
- Tuttavia come ci si comporta quando si vuole provare a rappresentare $\mathcal{Z}$ o $\mathcal{N}$ con un insieme **limitato** di elementi?
- Se infatti supponiamo di aver a disposizione solo **3 bit**, la linea dei numeri diventa un **segmento**
- Cosa accade quando "superiamo" i limiti del segmento?
- Possiamo elaborare un'aritmetica in grado di **operare coerentemente** su un segmento?



L'Aritmetica Modulare

- L'artimetica che "funziona" quando l'insieme di riferimento è finito, è l'**artimetica modulare**
- Essa fa uso dell'operazione di **modulo** che è definita come **resto della divisione intera**

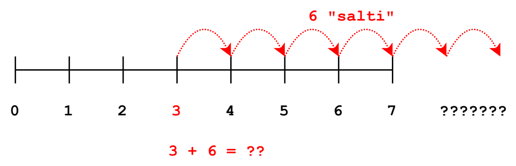
$$
\begin{aligned}
a \mod n &= \text{resto della divisione tra } a \text{ e } n \\
25 \mod 8 &= 1 \\
24 \mod 8 &= 0 \\
17 \mod 3 &= 2
\end{aligned}
$$

- Introduce il concetto di **congruenza modulo** n:

$$a \equiv b \pmod{n} \quad \Leftrightarrow \quad (a - b) \text{ multiplo di } n$$

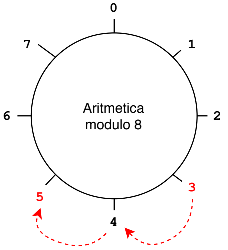
L'Aritmetica Modulare

- Nell'artimetica modulare, si definisce un **intero positivo** *n* che rappresenta la **dimensione del segmento** dei numeri
- In questo caso, abbiamo **n = 8**:

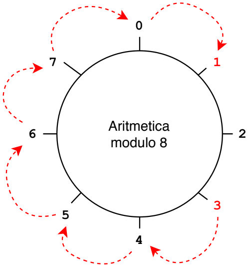


L'Aritmetica Modulare

- Il segmento si trasforma in una **circonferenza dei numeri**, andando a chiudere insieme le estremità
- Le operazioni si effettuano **spostandosi ciclicamente** lungo tale circonferenza
- Pertanto quando si supera il limite, si "**ricomincia da capo**" (dallo 0)



3 + 2 = 5



3 + 6 = 1

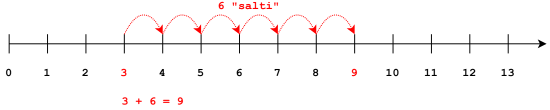
Aritmetica Modulare vs. Aritmetica tradizionale

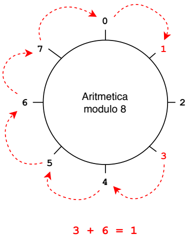
- Nell'esempio, i due risultati ottenuti con la stessa operazione **devono essere equivalenti**
- Poichè a destra abbiamo considerato l'artimetica modulare con $n = 8$ abbiamo che:

$$9 \equiv 1 \pmod{n}$$

Poichè:

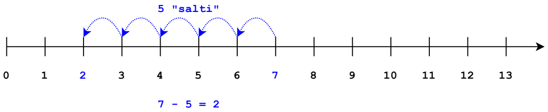
$$9 - 1 = 8 \text{ multiplo di } n$$

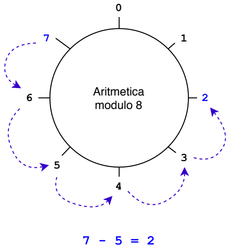




Aritmetica Modulare vs. Aritmetica tradizionale

- Anche la **sottrazione** opera allo stesso modo, utilizzando il concetto di "salto all'indietro"





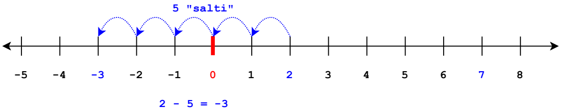
Aritmetica Modulare vs. Aritmetica tradizionale

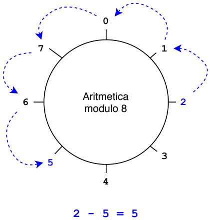
- Anche nella sottrazione, i risultati ottenuti con la stessa operazione nelle due aritmetiche **devono essere equivalenti**
- Poichè a destra abbiamo considerato l’artimetica modulare con $n = 8$ abbiamo che:

$$-3 \equiv 5 \pmod{n}$$

Poichè:

$$-3 - 5 = -8 \text{ multiplo di } n$$
$$5 - (-3) = 8 \text{ multiplo di } n$$





Rappresentazione in Complemento a 2

Complemento a 2

Rappresentazione in Complemento a 2

- Il complemento a 2 permette di rappresentare un **sottoinsieme dell'insieme** $\mathcal{Z}$ (numeri relativi) in un calcolatore, e di operare all'interno di tale sottoinsieme
- Dati k bit, con essi è possibile rappresentare l'intervallo:

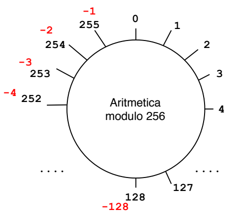
$$[-2^{k-1}, 2^{k-1} - 1] \subset \mathcal{Z}$$

il cui numero di elementi è:

$$2^{k-1} - 1 - (-2^{k-1}) + 1 = 2^{k-1} + 2^{k-1} = 2 \cdot 2^{k-1} = 2^k$$

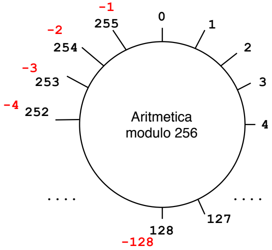
- Il **bit più significativo** (più a sinistra) permette di distinguere i positivi dai negativi:
  - Se esso è 0 stiamo rappresentando un numero **positivo**
  - Se esso è 1 stiamo rappresentando un numero **negativo**
- Le operazioni fanno uso dell'aritmetica modulare, con modulo $n = 2^k$

Rappresentazione in Complemento a 2 con 8 bit



Binario	Decimale	Numero rappresentato
1000 0000	128	-128
...	...	...
1111 1101	253	-3
1111 1110	254	-2
1111 1111	255	-1
0000 0000	0	0
0000 0001	1	+1
0000 0010	2	+2
...	...	...
0111 111	127	+127

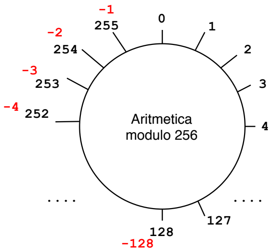
Operazioni in Complemento a 2



Somma Binaria in Complemento a 2 con 8 bit

Numero da Rappresentare	Decimale	Binario	
		1 1111 1000	(riporti)
-2 +	254	1111 1110	+
5 =	5	0000 0101	=
3	3	0000 0011	

Operazioni in Complemento a 2



Sottrazione Binaria in Complemento a 2 con 8 bit

$$5 - 3 = 5 + (-3) = 2$$

Numero da Rappresentare	Decimale	Binario	
		1 1111 1010	(riporti)
5 =	5	0000 0101	+
-3 +	253	1111 1101	=
2	2	0000 0010	

Operazioni in Complemento a 2

Cambio di Segno in Complemento a 2

1. Sia x un numero intero rappresentato in binario in complemento a 2 a k bit
2. Invertire **tutti i k bit** (da 0 a 1 e viceversa)
3. Sommare (artimeticamente) 1
4. Troncare i bit in eccesso
5. Il risultato sarà la rappresentazione di $-x$

Cambio di Segno in Complemento a 2

Numero	Binario	
5	0000 0101	inversione
	1111 1010 +	somma di 1
	1 =	
-5	1111 1011	

Cambio di Segno in Complemento a 2

Numero	Binario	
5	0000 0101	inversione
	1111 1010 +	somma di 1
	1 =	
-5	1111 1011	

Verifica

	1 1111 111	
5 +	0000 0101	+
-5 =	1111 1011	=
0	0000 0000	

Problemi del Complemento a 2

Dati k bit ...

1. L'insieme dei positivi e dei negativi **non hanno lo stesso numero di elementi**:
   - Positivi $[1, 2^{k-1} - 1]$, numero di elementi = $2^{k-1} - 1$
   - Negativi $[-2^{k-1}, -1]$,numero di elementi = 2^{k-1}
2. In caso di **overflow** il risultato diventerebbe **negativo**, ma comunque *corretto*

$$125 + 10 = 135(???)$$

	1111	(riporti)
125 +	0111 1101	+
10 =	0000 1010	=
-121	1000 0111	

Rappresentazione interi

Rappresentazione senza segno

Bit	Intervallo
8	[0, 255]
16	[0, 65535]
32	[0, 4294967295]

Rappresentazione in Complemento a 2

Bit	Intervallo
8	[-128, +127]
16	[-32768, +32767]
32	[-2147483648, 2147483647]

Rappresentazione a Virgola Fissa e a Virgola Mobile

Rappresentazione dei Reali

- Così come per l'insieme $\mathcal{Z}$, anche per l'insieme $\mathcal{R}$ avere un numero di bit **limitato** implica la possibilità di rappresentare solo un **sottoinsieme di $\mathcal{R}$**
- Pertanto, stabilito un sistema di rappresentazione, ogni numero di $\mathcal{R}$ verrà **approssimato** al valore più vicino rappresentabile
- Le tecniche di rappresentazione dei reali sono:
  - **virgola fissa**
  - **virgola mobile**

Rappresentazione in Virgola Fissa (Fixed Point)

- Fissato un numero di bit k, si stabilisce, all'interno del pattern di bit, una **posizione prefissata** della virgola
- In altri termini, si sceglie un numero r di bit da usare per rappresentare la **parte frazionaria** di un numero
- La parte intera sarà dunque rappresentata da $k - r$ bit
- Le cifre dopo la virgola avranno dunque **peso** (da sinistra verso destra) 2^{-1}, 2^{-2}, 2^{-3}, ..., in accordo con l'aritmetica dei sistemi di numerazione

Fixed Point, $k = 8$, $r = 4$

$$
\begin{aligned}
00110110 = 0011.0110 &= 2^1 + 2^0 + 2^{-2} + 2^{-3} \\
&= 3 + 0.25 + 0.125 \\
&= 3.375
\end{aligned}
$$

Rappresentazione dei Reali

Fixed Point: Da Reale a Binario

- Per convertire un numero da reale a binario la procedura è la seguente
- La **parte intera** si converte usando le divisioni successive
- Per **parte frazionaria** si segue l'algoritmo:
  1. Si moltiplica il numero per 2
  2. Si estrae la parte intera del risultato, che sarà la cifra 0 o 1
  3. Si estrae decimale del risultato e, se non è **zero**, si va al passo 1
  4. Le parti intere estratte, ricopiate da sinistra verso destra, costituiranno le **cifre decimali**

Fixed Point, $k = 8$, $r = 4$

3.25	3	1	0.25	0.5	0	$(0011.0100)_2$
	1	1	0.5	1.0	1	
	0		0			

Fixed Point, $k = 8$, $r = 4$

$$3.6 \qquad \begin{array}{c|c} 3 & 1 \\ 1 & 1 \\ 0 & \end{array} \qquad \begin{array}{c|c|c} 0.6 & 1.2 & 1 \\ 0.2 & 0.4 & 0 \\ 0.4 & 0.8 & 0 \\ 0.8 & 1.6 & 1 \\ 0.6 & \ldots & \ldots \end{array} \qquad (0011.1001)_2$$

Fixed Point, $k = 8$, $r = 4$

$$\begin{aligned} 00111001 = 0011.1001 &= 2^1 + 2^0 + 2^{-1} + 2^{-4} \\ &= 3 + 0.5 + 0.0625 \\ &= 3.5625 \end{aligned}$$

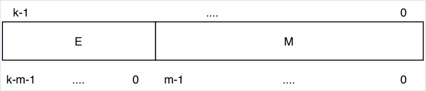
Problemi della Rappresentazione in Virgola Fissa

- La rappresentazione in virgola fissa ha lo svantaggio di poter rappresentare un sottoinsieme molto piccolo di $\mathcal{R}$
- Data una certa applicazione, la dimensione della parte frazionaria dipende molto dalle grandezze che occorre rappresentare, pertanto non esiste una scelta univoca
- La rappresentazione in **virgola mobile** consente una maggiore flessibilità a fronte dello stesso numero (limitato) di bit

Rappresentazione dei Reali

Rappresentazione in Virgola Mobile (Floating Point)

- Fissato un numero di bit k, si stabiliscono:
  - m bit di **mantissa**
  - $k - m$ bit di **esponente**



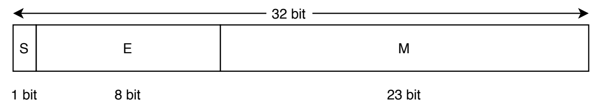
$$valore = M \cdot 2^E$$

- Il numero di bit m e la codifica di mantissa e esponente dipendono poi dallo **standard** adottato

Rappresentazione dei Reali

Floating Point IEEE 754

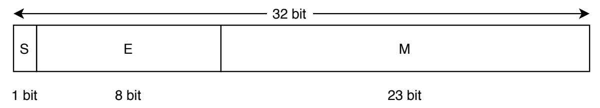
- L'attuale standard utilizzato per la virgola mobile è denominato **IEEE 754**
- E' lo standard usato dai tipi **`float`** del linguaggio C/C++ (e di tutti gli altri linguaggi di programmazione)
- Usa un'estensione di 32 bit con la seguente suddivisione:
  - **1 bit** per il **segno** $(0 = +, 1 = -)$
  - **8 bit** per l'**esponente** (con codifica *a eccesso 127*)
  - **23 bit** per la **parte frazionaria** della mantissa, la quale ha sempre la parte intera pari a 1 (di default)



$$(-1)^S \times 1.M \times 2^{E-127}$$

Rappresentazione dei Reali

Esempio: Floating Point IEEE 754



$$(-1)^S \times 1.M \times 2^{E-127}$$

$$(BF\ A0\ 00\ 00)_{16}$$

$$1011\ 1111\ \ 1010\ 0000\ \ 0000\ 0000\ \ 0000\ 0000$$

$$\boxed{1}\boxed{01111111}\boxed{01000000000000000000000}$$

$$(-1)^1 \times (1.01)_2 \times 2^{127-127}$$

$$-(1.01)_2 = -1.25$$

Rappresentazione dei Reali

Esempio: Floating Point IEEE 754

32 bit

S	E	M
1 bit	8 bit	23 bit

$$(-1)^S \times 1.M \times 2^{E-127}$$

$$(C0\ 53\ 33\ 33)_{16}$$

$$1100\ 0000\quad 0101\ 0011\quad 0011\ 0011\quad 0011\ 0011$$

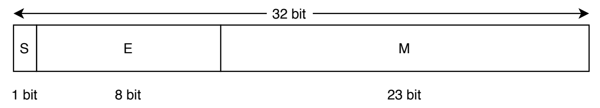
$$\boxed{1}\ \boxed{10000000}\ \boxed{10100110011001100110011}$$

$$(-1)^1 \times (1.10100110011001100110011)_2 \times 2^{128-127}$$

$$-(1.10100110011001100110011)_2 \times 2$$

$$-(11.0100110011001100110011)_2 \cong -3.3$$

Esempio: Floating Point IEEE 754



$$(-1)^S \times 1.M \times 2^{E-127}$$

$$(49\ 74\ 24\ 00)_{16}$$

$$0100\ 1001\quad 0111\ 0100\quad 0010\ 0100\quad 0000\ 0000$$

$$\boxed{0}\boxed{10010010}\boxed{11101000010010000000000}$$

$$(1.11101000010010000000000)_2 \times 2^{146-127}$$

$$(1.11101000010010000000000)_2 \times 2^{19}$$

$$(11110100001001000000.0000)_2 = 1000000$$

Rappresentazione dei Caratteri

Rappresentazione dei Caratteri

- Come ogni tipologia di informazione, anche il **testo** necessita di un sistema di rappresentazione “in bit”
- L’obiettivo è consentire la manipolazione di testo da parte di un calcolatore, il suo inserimento da una tastiera, la sua visualizzazione e la stampa
- Nel tempo, sono stati elaborati diversi standard di rappresentazione dei caratteri:
  - **EBCDIC** Extended Binary Coded Decimal Interchange Code *Standard europeo non più usato*
  - **ASCII** American Standard Code for Information Interchange *Standard attualmente in uso*
- Tali “standard” forniscono delle tabelle di corrispondenza tra le *lettere, i simboli di interpunzione, e altri simboli* con un equivalente **valore numerico a *n* bit**

Rappresentazione dei Caratteri

Codifica ASCII Base

- Codice a **7 bit** (in un byte il l'ultimo bit è dunque sempre a 0)
- Codifica i caratteri e simboli dell'alfabeto americano, più alcuni *"caratteri di controllo"*

USASCII code chart

$b_7\ b_6\ b_5$					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Row ↓ / Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

(Fonte: Wikipedia)

Codifica ASCII Base: Lettere

- **Lettere Maiuscole:** Da hex **41** a hex **5A**
- **Lettere Minuscole:** Da hex **61** a hex **7A**

USASCII code chart

b_7 b_6 b_5					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b_4	b_3	b_2	b_1	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

(Fonte: Wikipedia)

Codifica ASCII Base: Simboli

- Da hex **20** a hex **40**
- Da hex **5B** a hex **60**
- Da hex **7B** a hex **7F**

USASCII code chart

b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

(Fonte: Wikipedia)

Codifica ASCII Base: Simboli

- Da hex **20** a hex **40**
- Da hex **5B** a hex **60**
- Da hex **7B** a hex **7F**

USASCII code chart

$b_7 b_6 b_5$ →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

(Fonte: Wikipedia)

Rappresentazione dei Caratteri

Codifica ASCII Base: Caratteri di Controllo

- Da hex **00** a hex **1F**
- Non producono *simboli stampabili* ma **azioni specifiche** sullo schermo
- Es.: 0D (Carriage Return) cursore a capo
- 0A (Line Feed) cursore alla linea successiva
- 08 (Back Space) cursore al carattere precedente

USASCII code chart

b_7 b_6 b_5 →					000	001	010	011	100	101	110	111
b_4	b_3	b_2	b_1	Row ↓ / Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Rappresentazione dei Caratteri

Codifica ASCII Estesa

- Codice a **8 bit**
- Il set hex $[0, 7F]$ corrisponde all'ASCII base
- Il set hex $[80, FF]$ include simboli grafici e caratteri di alfabeti internazionali (europei)

Extended ASCII Chart (character codes 128 – 255)

128	Ç	143	Å	158	₧	172	¼	186	║	200	╚	214	╓	228	Σ	242	≥
129	ü	144	É	159	ƒ	173	¡	187	╗	201	╔	215	╫	229	σ	243	≤
130	é	145	æ	160	á	174	«	188	╝	202	╩	216	╪	230	µ	244	⌠
131	â	146	Æ	161	í	175	»	189	╜	203	╦	217	┘	231	τ	245	⌡
132	ä	147	ô	162	ó	176	░	190	╛	204	╠	218	┌	232	Φ	246	÷
133	à	148	ö	163	ú	177	▒	191	┐	205	═	219	█	233	Θ	247	≈
134	å	149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Ω	248	°
135	ç	150	û	165	Ñ	179	│	193	┴	207	╧	221	▌	235	δ	249	∙
136	ê	151	ù	166	ª	180	┤	194	┬	208	╨	222	▐	236	∞	250	·
137	ë	152	ÿ	167	º	181	╡	195	├	209	╤	223	▀	237	φ	251	√
138	è	153	Ö	168	¿	182	╢	196	─	210	╥	224	α	238	ε	252	ⁿ
139	ï	154	Ü	169	⌐	183	╖	197	┼	211	╙	225	ß	239	∩	253	²
140	î	155	¢	170	¬	184	╕	198	╞	212	╘	226	Γ	240	≡	254	■
141	ì	156	£	171	½	185	╣	199	╟	213	╒	227	π	241	±	255	
142	Ä	157	¥														

UNICODE

- Codifica originariamente **16 bit**, poi estesa a **21 bit**
- Include l'ASCII e aggiunge la codifica per lettere e simboli di tutti gli alfabeti internazionali
- Tuttavia presenta delle incompatibilità con l'ASCII in quanto quest'ultimo usa solo 8 bit $\Rightarrow$ un file codificato in UNICODE non sarebbe interpretato correttamente da un software che comprende solo l'ASCII

UTF-8

- **Unicode Translation Format, 8 bit** è un formato di traduzione dell'UNICODE su un **bit size variabile**
- Consente di superare i problemi di incompatibilià tra UNICODE e ASCII

UNICODE	UTF-8	
0x00 - 0x7F	`0xxx xxxx`	Codifica ASCII base su 7 bit
0x80 - 0x7FF	`110x xxxx 10xx xxxx`	
0x800 - 0xFFFF	`1110 xxxx 10xx xxxx 10xx xxxx`	

Rappresentazione dell’Informazione

Corrado Santoro

Dipartimento di Matematica e Informatica
santoro@dmi.unict.it

Corso di Architettura degli Elaboratori