

3 – Rappresentazione e Codifica delle Informazioni

- Sistemi di Numerazione
- Codifica dell'Informazione
- Aritmetica Binaria

1

3 – Rappresentazione e Codifica delle Informazioni

Sistemi di Numerazione

Richiami

- *Sistema di Numerazione Posizionale*
 - Il valore di un numero dipende dalla base adottata, dalle cifre che lo compongono, dalla loro posizione e dall'eventuale segno (+/-)
 - Ad ogni cifra del numero è associato un peso in base alla posizione
 - Elementi caratteristici
 - **Base**
 - Elemento fondamentale che caratterizza il sistema di numerazione
 - **Cifre**
 - Usate per comporre i numeri: da 0 a *Base-1*
 - **Posizione**
 - La posizione della cifra (la più a destra è 0) rappresenta l'esponente da assegnare alla base per calcolare il peso della cifra in una data posizione
 - Sistemi di Numerazione NON Posizionali
 - Es. Numeri Romani

3

Richiami

- Sistema Decimale
 - Base: 10
 - Cifre: 0-9
 - Posizione: esponente
 - *Esempio*
 - $1215 = 1 \cdot 10^3 + 2 \cdot 10^2 + 1 \cdot 10^1 + 5 \cdot 10^0$
- Sistema Binario
 - Cifre: 0-1
 - Base: 2
 - Posizione: esponente
 - *Esempio*
 - $1101 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

4

Richiami

- Sistema Esadecimale
 - Base: 16
 - Cifre: 0-9 A B C D E F
 - Posizione: esponente
 - *Esempio*
 - $1BA2 = 1 \cdot 16^3 + 11 \cdot 16^2 + 10 \cdot 16^1 + 2 \cdot 16^0$
- Sistema Ottale
 - Base: 8
 - Cifre: 0-7
 - Posizione: esponente
 - *Esempio*
 - $17 = 1 \cdot 8^1 + 7 \cdot 8^0$

5

Cambiamenti di Base

- Da Base b a base 10
 - Scomporre il modulo del numero da trasformare nella sommatoria delle cifre (trasformate singolarmente in *base 10*) moltiplicate per il loro peso
 - Effettuare tale somma in base 10 e sistemare l'eventuale segno
- Esempi
 - Come in pratica già visto
 - $1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 1 = 13_{10}$
 - $1BA2_{16} = 1 \cdot 16^3 + 11 \cdot 16^2 + 10 \cdot 16^1 + 2 \cdot 16^0 = 7074_{10}$
 - $-17_8 \Rightarrow 17_8 = 1 \cdot 8^1 + 7 \cdot 8^0 = 15_{10} \Rightarrow -15_{10}$

6

Cambiamenti di Base

- Da *base 10* a base *b*
 - Metodo delle divisioni successive
 - Si divide il modulo del valore decimale per *b* fino ad ottenere 0 e si conserva il resto
 - Si scrivono i resti a partire dall'ultimo ottenuto trasformandoli in cifre della nuova base e infine si sistema il segno

- Esempi

123₁₀ in base 8?

123/8 = 15 con resto 3
15/8 = 1 con resto 7
1/8 = 0 con resto 1

Risultato: 123₁₀ = 173₈

-123₁₀ in base 16?

123/16 = 7 con resto 11
7/16 = 0 con resto 7
11 corrisponde ad B

Risultato: -123₁₀ = -7B₁₆

7

3 – Rappresentazione e Codifica delle Informazioni

Codifica delle Informazioni

Codifica delle Informazioni

- In un calcolatore, i dati (e le istruzioni) sono codificati in forma binaria, ovvero come sequenze finite di cifre 0 e 1
- Il **bit** è la più piccola unità di informazione in un calcolatore
 - Un bit può avere valore 0 oppure 1
 - La parola bit è una forma contratta per *binary digit* (cifra binaria)
- Ciascun bit è memorizzato da una cella elementare di memoria, fisicamente realizzata come dispositivo elettronico in cui sono chiaramente distinguibili due stati
 - Questi due stati vengono fatti corrispondere allo 0 e all'1

9

Codifica delle Informazioni

- In genere il bit è una unità di informazione troppo piccola per essere usata individualmente
 - I dati vengono codificati sotto forma di **sequenze di bit**
 - Ciascuna sequenza di bit può essere interpretata come un dato solo se viene opportunamente decodificata
- Un'altra importante unità di informazione è il byte
 - Un **byte** è una sequenza di 8 bit

10

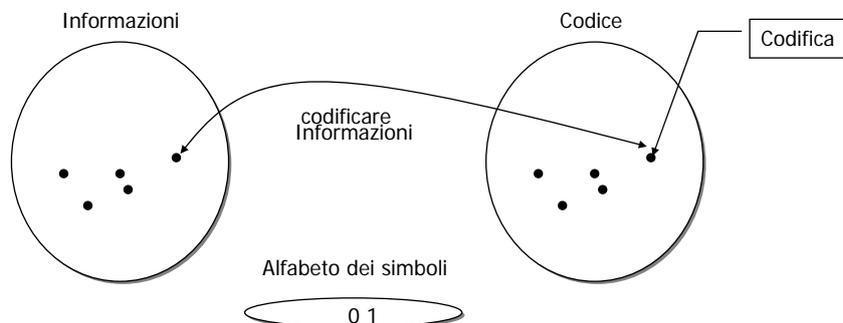
Codifica delle Informazioni

- Per consentire la la corretta interpretazione delle sequenze binarie, vengono utilizzati dei meccanismi di *tipizzazione*
- Tipo di dato elementare
 - Alfabetico
 - Caratteri dell'alfabeto ASCII (8 bit)
 - Caratteri dell'alfabeto Unicode (16 bit)
 - Numerico
 - Interi relativi (ovvero, con segno) a 32 bit
 - Numeri razionali in virgola mobile secondo lo standard IEEE 754-1985 a 32 bit
 - 9 cifre significative e mantissa tra -45 e +38
 - Logico o *Booleano*
 - Valori logici: vero o falso (1 bit)

11

Codifica binaria

- Con n bit si ottengono 2^n codici diversi
 - Per codificare n elementi servono $\lceil \log_2(n) \rceil$ bit



12

Codifica binaria per dati di tipo alfanumerico

- Codici alfanumerici
 - Associano ad ogni carattere una sequenza di bit.
 - Si utilizzano degli standard di conversione:
 - Codice ASCII:
 - Utilizza una sequenza di 8 bit
 - » Es. Il carattere '0' è identificato da 00110000.
 - Codice UNICODE
 - Utilizza una sequenza di 16 bit.

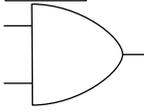
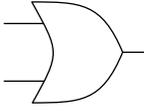
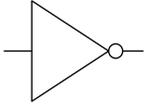
13

Codifica binaria per i dati di tipo *Booleano* o Logico

- Un valore logico è associato ad un solo bit
 - Tipicamente
 - VERO → 1
 - FALSO → 0
- Operazioni definite sui dati di tipo logico
 - **AND**
 - **OR**
 - **NOT**
- I dispositivi fisici realizzano in modo molto semplice queste operazioni.

14

Operazioni definite per i dati di tipo *Booleano* o Logico

AND	Ingressi	Risultato	Simboli
	0 0	0	
	0 1	0	
	1 0	0	
	1 1	1	
OR	Ingressi	Risultato	
	0 0	0	
	0 1	1	
	1 0	1	
	1 1	1	
NOT	Ingresso	Risultato	
	0	1	
	1	0	

15

Codifica binaria per i numeri naturali (*Binario Puro*)

- Si usa il *Sistema di Numerazione Binario*

- Con n bit si codificano i numeri da 0 a $2^n - 1$

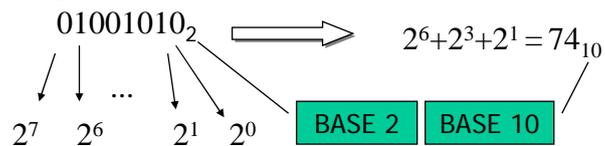
- $N = a_{n-1} a_{n-2} a_{n-3} \dots a_0$

- dove a_i vale 0 oppure 1

- Ogni simbolo a_i nella sequenza è associato ad un peso 2^i

- Valore di N in base 10 è $a_{n-1} * 2^{n-1} + a_{n-2} * 2^{n-2} + \dots + a_0$

- Esempio: **conversione da base 2 a base 10**



16

Codifica binaria per i numeri naturali (*Binario Puro*)

- Conversione da base 10 a base 2
 - Metodo delle divisioni successive
 - Si divide il valore decimale per 2 fino ad ottenere 0 e si conserva il resto
 - Si scrivono i resti a partire dall'ultimo ottenuto
 - Esempio:

	$44_{10}:2$	$22_{10}:2$	$11_{10}:2$	$5_{10}:2$	$2_{10}:2$	$1_{10}:2$	0_{10}
Resto	0	0	1	1	0	1	

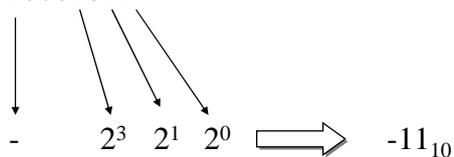


Numero binario: **101100₂**

17

Codifica binaria per i numeri interi

- Codifica **Modulo e Segno** con n bit
 - Un bit, quello a sinistra, rappresenta il segno
 - Convenzione: 0 per +, 1 per –
 - Gli altri $n-1$ bit sono riservati al modulo del numero
 - Con n bit si codificano i numeri da $-2^{n-1}-1$ a $2^{n-1}-1$
- Esempio: 10001011



18

Codifica binaria per i numeri interi

- Codifica **Complemento a 2** con n bit
 - **Positivi:** il primo bit è 0 gli altri codificano il valore come nella rappresentazione *modulo e segno*
 - **Negativi:** si considera il valore positivo corrispondente e si codifica (il primo bit alla fine è sempre 1)
 - *Procedura di codifica*
 - A partire dal bit più a destra si lascia ogni bit inalterato fino al primo 1 incluso
 - Dal bit successivo si cambia ogni 1 in 0 ed ogni 0 in 1
 - Con n bit si codificano i numeri da -2^{n-1} a $2^{n-1}-1$

19

Codifica binaria per i numeri interi

- Codifica in complemento a 2
 - **Esempio 1: $+12_{10}$ su 6 bit**
 - 0 per il segno e 01100 per il modulo
 - Il complemento a 2 di $+12$ è **001100**
 - **Esempio 2: -12_{10} su 6 bit**
 - Prima consideriamo $+12$ in complemento a 2 su 6 bit
 - Come già visto questo è 001100
 - Poi si applica la *procedura di codifica*
 - Da destra verso sinistra prendiamo i bit fino al primo 1: ---100
 - Poi invertiamo (0 diventa 1 e viceversa): 110100
 - Il complemento a 2 di -12 è **110100**
 - N.B. La procedura di codifica ci permette di cambiare il segno
 - » Riapplicandola si torna al C2 di $+12$!

20

Codifica binaria per i numeri interi

- Relazione tra informazione numerica e codice

	<i>Binario Puro</i>	<i>Modulo e Segno</i>	<i>Complemento a 2</i>
000	0	+0	+0
001	1	+1	+1
010	2	+2	+2
011	3	+3	+3
100	4	-0	-4
101	5	-1	-3
110	6	-2	-2
111	7	-3	-1

21

Codifica binaria per i numeri con parte frazionaria

- Rappresentazione in virgola mobile
 - È utilizzata come modello dei numeri reali
 - I numeri sono rappresentati secondo la notazione scientifica $n=m \cdot 2^e$ dove m è la mantissa ed e l'esponente.
 - *Standard IEEE 754*
 - La rappresentazione della mantissa è in *modulo e segno*
 - Precisione singola: 32 bit
 - » un bit di segno, 8 bit di esponente, 23 bit di mantissa
 - Doppia precisione: 64 bit
 - » un bit di segno, 11 bit di esponente, 52 bit di mantissa
 - Precisione estesa: 80 bit
 - » un bit di segno, 15 bit di esponente, 64 bit di mantissa

22

Cambiamenti di base da/verso C2

- **Da base 10 a C2**

- Si considera prima il modulo del numero da trasformare
- Si trasforma tale modulo in binario puro tramite il metodo delle divisioni successive
- Si passa al C2 del modulo
 - Deve iniziare con uno 0 in quanto positivo!
- Se il numero di partenza era positivo resta uguale altrimenti si inverte applicando l'opportuna regola

23

Cambiamenti di base da/verso C2

- **Esempio 1**

- Convertire il numero 1215_{10} in C2
- Essendo il numero positivo il modulo è uguale: 1215
- Il binario puro è 1001011111_2
- Per passare in C2 si aggiunge uno 0: 01001011111_{C2}
- il numero di partenza era positivo 01001011111_{C2}

- **Esempio2**

- Convertire il numero -178_{10} in C2
- Convertiamo il modulo in binario puro: 10110010_2
- C2 del modulo: 010110010_{C2}
- Il numero di partenza era negativo: invertiamo $\Rightarrow 101001110_{C2}$
 - Occhio all'uno davanti!!!

24

Cambiamenti di base da/verso C2

- **Da C2 a base 10**

- Una volta ottenuto il modulo è una semplice somma di prodotti.

Quindi:

- Se il numero è negativo invertirlo (e ricordarselo!)
- Somma di prodotti e aggiustamento del segno
- Esempio 1
 - Convertire il numero **01001011111**_{C2} in base 10
 - Il numero è positivo quindi va bene così com'è
 - Somma di prodotti
 - » $1 \cdot 2^{10} + 1 \cdot 2^7 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1215_{10}$ (fine!)
- Esempio 2
 - Convertire il numero **10010**_{C2} in base 10.
 - E' negativo: invertito => 01110
 - Somma di prodotti: $0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 14$
 - Rimettere il segno: **-14**₁₀

25

Cambiamenti di base da/verso C2

- **Da base 16 a C2 e viceversa**

- Essendo la base una potenza della base due, la conversione del modulo in/dal binario puro è molto semplice
- In particolare essendo $16=2^4$ basta esprimere ogni cifra del numero esadecimale in binario su 4 bit
- Al contrario basta esprimere i bit, presi 4 alla volta da destra (meno significativo) verso sinistra, con la corrispondente cifra esadecimale
- Dal binario puro procedere come già noto

- **Da base 8 a C2 e viceversa**

- Essendo $8=2^3$ il procedimento è analogo alla base 16 lavorando però con gruppi di 3 bit

26

Cambiamenti di base da/verso C2

- Esempio 1
 - Esprimere il numero $-E1A_{16}$ in C2
 - Tralasciando il segno: E_{16} è 1110_2 , 1_{16} è 0001_2 , A_{16} è 1010_2 quindi la conversione in binario puro è 111000011010_2
 - Nota: togliere gli eventuali zeri a sinistra!!!
 - Passare in C2 aggiungendo uno 0 davanti: 0111000011010_{C2}
 - Il numero era negativo, invertire: 1000111100110_{C2}
- Esempio 2
 - Esprimere il numero 10110101_{C2} in esadecimale
 - Passiamo al positivo: 01001011
 - Partendo da destra verso sinistra 1011_2 è B_{16} , 0100_2 è 4_{16} quindi la conversione è $4B_{16}$
 - Rimettiamo il segno: $-4B_{16}$

27

Cambiamenti di base da/verso C2

- **Da Modulo e Segno a C2 e viceversa**
 - In base al segno passare al modulo e poi trasformare in modo opportuno
 - Esempio 1
 - Esprimere il numero 11001_{MS} in C2
 - Modulo: 1001
 - Modulo in C2: 01001
 - Consideriamo il segno - invertendo: 10111_{C2}
 - Esempio 2
 - Esprimere il numero 11001_{C2} in MS (su 5 bit)
 - Prendiamo il modulo: 00111
 - Modulo in binario puro: 0111 (su 4 bit!)
 - Rimettiamo il segno: 10111_{MS}

28

3 – Rappresentazione e Codifica delle Informazioni

Aritmetica Binaria

Aritmetica Binaria in C2

- Il complemento a 2 include il segno nella codifica
- Questo semplifica il modo di fare le operazioni
 - L'operazione $A_{C2} + B_{C2}$ si svolge normalmente
 - L'operazione $A_{C2} - B_{C2}$ può essere vista come $A_{C2} + (-B_{C2})$
 - Quindi le sottrazioni si riducono alla **somma** del complemento a 2 di A con il complemento a 2 di -B
 - Se abbiamo B_{C2} sappiamo come cambiargli il segno!
 - *Procedura di codifica*
 - L'operazione $-A_{C2} - B_{C2}$ si trasforma in $-A_{C2} + (-B_{C2})$
 - Cioè nella **somma** del complemento a 2 di -A con il complemento a 2 di -B
 - Ci basta cambiare segno ad A_{C2} e B_{C2} e sommare!

Operazioni tra numeri interi

- Prospetto di base

Operandi	Riporto precedente	
	0	1
0 + 0	0 riporto 0	1 riporto 0
0 + 1	1 riporto 0	0 riporto 1
1 + 0	1 riporto 0	0 riporto 1
1 + 1	0 riporto 1	1 riporto 1

- Nessun bit a 1: risultato 0, riporto 0
- Un bit a 1: risultato 1, riporto 0
- Due bit a 1: risultato 0, riporto 1
- Tre bit a 1: risultato 1, riporto 1

31

Operazioni in C2 tra numeri interi

- Es. Operazione $A_{C2} + B_{C2}$

- $A = 39_{10}$

- $B = 14_{10}$

Riporto	0	0	0	1	1	1	0	-
A_{C2}	0	1	0	0	1	1	1	
B_{C2}	0	0	0	1	1	1	0	
Risultato $_{C2}$	0	1	1	0	1	0	1	

+ 39_{10}
 14_{10}
 = 53_{10}

32

Operazioni in C2 tra numeri interi

- Operazione $A_{C2} - B_{C2} = A_{C2} + (-B_{C2})$

B_{C2}	0	0	0	1	1	1	0
$-B_{C2}$	1	1	1	0	0	1	0

Riporto	1	1	0	0	1	1	0	-
A_{C2}	0	1	0	0	1	1	1	39_{10}
$-B_{C2}$	1	1	1	0	0	1	0	-14_{10}
Risultato $_{C2}$	0	0	1	1	0	0	1	25_{10}

Operazioni in C2 tra numeri interi

- Operazione $-A_{C2} - B_{C2} = (-A_{C2}) + (-B_{C2}) = -(A_{C2} + B_{C2})$

B_{C2}	0	0	0	1	1	1	0
$-B_{C2}$	1	1	1	0	0	1	0
A_{C2}	0	1	0	0	1	1	1
$-A_{C2}$	1	0	1	1	0	0	1

Riporto	1	1	1	0	0	0	0	-
$-A_{C2}$	1	0	1	1	0	0	1	-39_{10}
$-B_{C2}$	1	1	1	0	0	1	0	-14_{10}
Risultato $_{C2}$	1	0	0	1	0	1	1	-53_{10}
	0	1	1	0	1	0	1	35_{10}

Operazioni in C2 tra numeri interi

- Overflow
 - Quando si lavora con un numero di fissato bit è possibile che il risultato di alcune operazioni possa non essere rappresentabile in tale numero di bit
 - In tal caso si parla di *overflow*
 - Es. $111+111=1000$
 - Se fossi costretto ad usare solo 3 bit vedrei come risultato 000 che in questo caso è sbagliato
 - In questo caso quindi l'ultimo riporto è importante!
 - E' allora necessario saper riconoscere quando l'ultimo riporto è fondamentale per la correttezza del risultato
 - Bisogna saper riconoscere i casi di overflow

35

Operazioni in C2 tra numeri interi

- Riconoscere l'overflow
 - Riconoscere i casi di overflow è semplice soprattutto pensando a cosa può accadere
 - Operazione di somma in C2
 - **Se si sommano due numeri con segno diverso l'overflow non può mai accadere in quanto il risultato in modulo sarà sempre minore di uno dei due operandi**
 - » Dato che gli operandi erano rappresentabili sul numero di bit fissato lo sarà anche il risultato!
 - **Se si sommano due numeri con segno uguale e il risultato è dello stesso segno vuol dire che non c'è overflow**
 - » Siamo ancora nell'intervallo rappresentabile
 - **Se si sommano due numeri con segno uguale e il risultato è di segno diverso evidentemente c'è qualcosa che non va!**
 - » **OVERFLOW!**

36