Appunti del Corso di

Architettura degli Elaboratori I

Prof. P. Velardi

Nota: questi appunti sono stati preparati con l'intenzione di aiutare lo studente a seguire le lezioni e a colmare lacune derivanti da eventuali assenze. Non sono sostitutivi dei libri di testo.

Per un elenco di libri di testo e link utili, vedere la pagina web del corso.

Per la preparazione degli appunti, sono stati consultati i seguenti testi:

F. Fummi, M.G. Sami, C. Silvano progettazione Digitale Mc Graw Hills
T. Bartee "Digital Computer Fundamentals" Mc Graw Hill
J. Hayes " Computer Architecture and Organization" Mc Graw Hill
P. Fioretti "Reti Logiche" appunti dalle lezioni - Facolt di Ingegneria di Ancona
M. Del Santo Reti Logiche http://www.ing.unisannio.it/disanto/Teaching/RetiLogiche/2004

Ringrazio gli studenti che hanno segnalato e segnalano errori (che continuano a sopravvivere!)

Gli appunti sono divisi in 3 parti:

Parte I. Concetti introduttivi. Brevi cenni storici sull'architettura degli elaboratori. Rappresentazione dell'informazione. Codici. Algebra Booleana.
Parte II. Reti combinatorie: analisi e sintesi.
Parte III. Reti sequenziali: analisi e sintesi.

Parte I :
Sistemi Digitali

1.1 Introduzione

DEF **Sistemi Digitali** Sono tutti quegli apparati al cui interno le grandezze fisiche impiegate come segnali sono vincolate ad assumere solo valori discreti.

DEF **Sistemi Binari** Sono quei sistemi digitali in cui i segnali sono limitati a due valori di regime.

Rispetto ai sistemi analogici, nei quali i segnali possono assumere tutti i possibili valori in un continuo, i sistemi digitali consentono una **minore complessit** dei dispositivi che devono generare i segnali, ed una maggiore **immunit ai disturbi**.

Un sistema digitale un circuito costituito da *componenti elementari* e dai *collegamenti* che li interconnettono. I componenti elementari possono essere di due tipi: **porte** o elementi di **memoria**.

L'obbiettivo di questo corso lo studio di problemi di analisi e sintesi di sistemi digitali.

¥ La risoluzione di un problema di **analisi** di un sistema digitale consiste nell'individuazione delle relazioni di causa/effetto tra i segnali di ingresso e uscita di un circuito digitale, attraverso l'esame di una rappresentazione schematica dei suoi componenti elementari e dei collegamenti che li interconnettono.
¥ La risoluzione di un problema di **sintesi** consiste nella selezione dei circuiti digitali e delle interconnessioni necessarie per realizzare un nuovo e pi complesso circuito digitale, di cui pre-assegnata la **specifica funzionale**.

Viste della progettazione digitale.
Vista comportamentale: descrive le funzioni indipendentemente dall implementazione (es: progettare un circuito che esegua la somma aritmetica fra due numeri interi)
Vista strutturale: decsrive il modello di interconnessione dei componenti (es: disegno dei componenti digitali elementari e loro interconnessioni)
Vista fisica: componenti fisici (es. transistors, layout)

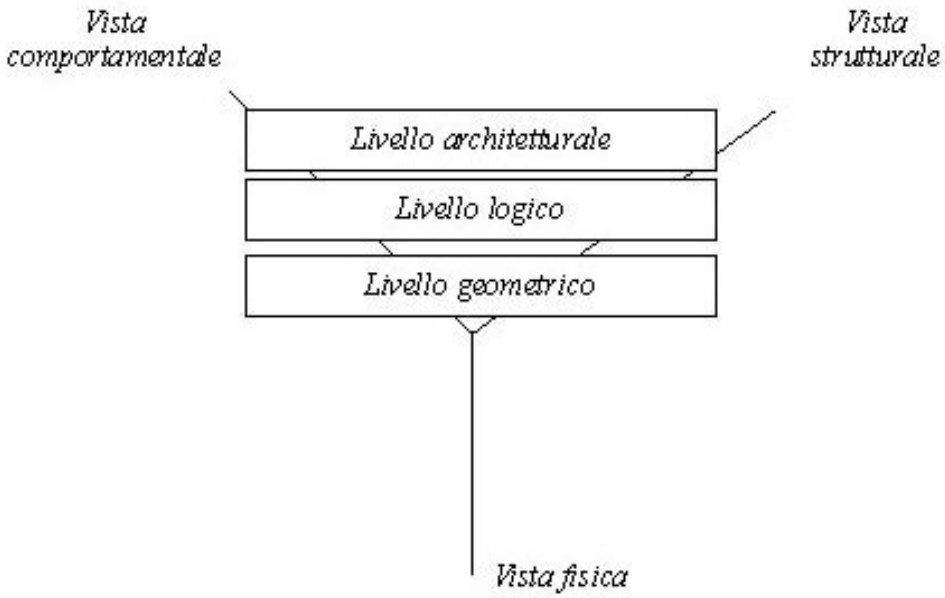


Figura 1.3 Diagramma a Y delle viste della progettazione digitale.

Campi di Applicazione

La progettazione digitale interessa tutti i campi di applicazione dell'elettronica:

Calcolo Automatico
Telecomunicazioni
Controlli Automatici
Misure Elettriche

....

In questo corso, oltre ad introdurre i principi generali di progetto di sistemi digitali, siamo interessati a studiare applicazioni nel campo del Calcolo Automatico.
Gli **Elaboratori Elettronici** sono sistemi digitali complessi in grado di leggere, elaborare e trasmettere informazioni binarie.

In Figura 1 sono mostrati i principali componenti di un Computer digitale.
Come si vede dalla figura, esistono 4 funzioni logiche fondamentali:
- **Controllo** : la sezione di controllo di un elaboratore preposta a sequenziare le operazioni di un computer, controllando le azioni di tutti gli altri moduli. I circuiti di controllo interpretano le istruzioni che costituiscono un programma e dirigono l'esecuzione delle medesime.
- **Elaborazione Logico-Aritmetica:** questa sezione in grado di eseguire operazioni aritmetiche e logiche (descritte in questo corso). L'unit di controllo indica quali operazioni eseguire e come.
- **Memoria**: questa sezione contiene l'informazione da utilizzare durante l'elaborazione. Per informazione si intendono istruzioni e dati. Distinguiamo due tipi di memoria: la memoria principale, realizzata con tecnologia digitale, risiede sull'elaboratore ed direttamente accessibile dai dispositivi di controllo; la memoria secondaria accessibile tramite opportune interfacce.
- **Ingresso-uscita**: abilita la lettura di dati all'interno della macchina ed il trasferimento di dati all'esterno. I dispositivi esterni possono essere:
- dispositivi di memorizzazione (dischi, nastri, ecc.)

- dispositivi di ingresso (sensori, tastiere, fax..)
- dispositivi di uscita (stampanti, video, fax..)

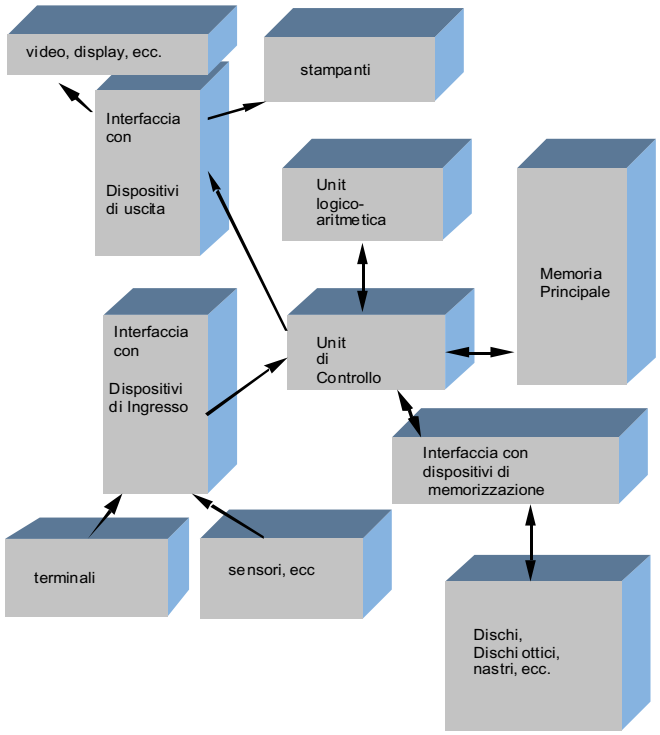


Figura 1.1 *Schema a Blocchi di un Computer Digitale*

Durante il suo funzionamento, un elaboratore cicla fra due stati, che complessivamente costituiscono il *ciclo di istruzione*:
Stato di Fetch: la macchina seleziona una istruzione eseguibile, e reperisce le informazioni necessarie all'esecuzione (operandi, tipo di operazione)
Stato di Execute: la macchina seleziona gli opportuni circuiti logico-aritmetici necessari ad eseguire l'elaborazione specificata nell'istruzione

La macchina sopra descritta - in termini sintetici - denominata Macchina di Von Neumann, e fu concepita nei suoi elementi logici fondamentali da John Von Neumann negli anni 40.
Nel paragrafo successivo studieremo brevemente la storia degli elaboratori, da Pascal, a Von Neumann, alle macchine dell'ultima generazione.

1.2 Storia degli Elaboratori da Pascal a Von Neumann

Gli studenti interessati possono consultare: H. Goldstine "Il Computer da Pascal a Von Neuman" ETAS libri..

1.2.1 Le prime macchine da calcolo: Pascal, Leibniz, Babbage

Il primo supporto conosciuto al calcolo manuale risale agli antichi Romani: l'*abacus*. I termini "calcolare" e "calcolatore" derivano dal latino "*calculi*", le file di sassolini di un abaco. Non fu che nel 1600 che furono realizzati supporti meno primitivi.
Whilhelm Schikard (1592-1635) Blaise Pascal (1623-1662) e Whilhelm Leibniz (1646-1716) costruirono macchine - simili nella sostanza- che consentivano di eseguire le 4 operazioni mediante un semplice meccanismo di ruote dentate.

Ad esempio nella macchina di Pascal, il meccanismo consisteva di due file di dischi combinatori dentati, collegati orizzontalmente e verticalmente. Ogni disco poteva essere ruotato di 10 posizioni: una rotazione completa causava, per trasmissione, la rotazione di una posizione del disco adiacente a sinistra, realizzando cos il meccanismo del riporto.
Leibniz dette tuttavia un contributo assai pi rilevante agli elaboratori, con i suoi studi di quella che ora conosciuta come la logica simbolica. Nel suo De Arte Combinatoria, descrisse il calcolo combinatorio come "un metodo generale nel quale tutte le verit della ragione dovrebbero essere ridotte ad una specie di calcolo".
Leibniz diede quattro grandi contributi nel campo del calcolo automatico:
- l'avvio della logica formale
- la costruzione di una macchina da calcolo
- la comprensione del carattere disumano del calcolo e l'opportunit , nonch la capacit , di automatizzare questo lavoro;
- l'idea che le macchine da calcolo potrebbero essere utilizzate per verificare le ipotesi

La tematica di Leibniz - come liberare gli uomini dalla schiavit di calcoli noiosi- fu ripresa un secolo dopo da Charles Babbage (1791-1871).
Babbage riusc ad ottenere un finanziamento dal governo britannico per realizzare la sua Macchina alle Differenze, una versione pi avanzata della macchina di Pascal, in grado di calcolare con il metodo delle differenze finite i valori di un polinomio di terzo grado.
Il teorema di Weierstrass afferma che ogni funzione continua in un intervallo finito pu essere approssimata tanto bene quanto si vuole da un polinomio. Perci il calcolo automatico dei valori di un polinomio in un certo intervallo riveste una certa importanza.
La Macchina alle Differenze poteva eseguire il calcolo dei valori di un polinomio attraverso una serie ripetuta di somme: una volta attivata ed impostata con i valori iniziali, la macchina proseguiva i suoi calcoli automaticamente, grazie ad un motore a vapore (la tecnologia di quei tempi!!). Babbage non complet la realizzazione del progetto della macchina, perch nel frattempo prese a lavorare ad un secondo prototipo: quello della Macchina Analitica. La Macchina Analitica merita qualche dettaglio perch - tecnologia a vapore a parte- presenta importanti similitudini con le moderne macchine elettroniche.
La Macchina Analitica il primo esempio di macchina a programma registrato: applicando al calcolo automatico l'idea della carta perforata ideato da Jaquard per automatizzare la realizzazione di orditi nell'industria tessile, Babbage dot la sua macchina della capacit di eseguire sequenze di calcoli preregistrate su schede perforate (il principio di funzionamento di consentire o meno il collegamento fra ingranaggi disponendo opportunamente dei fori sulle schede).

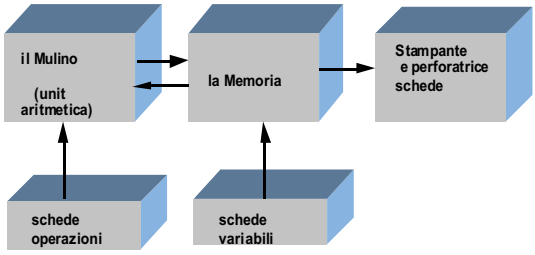


Figura 1.2 *La Macchina* ***Analitica di Babbage***

La Macchina Analitica consisteva di una unità di calcolo, detta Mulino (Mill) e di una memoria, costituita da una pila di registri in cui venivano memorizzati dati e risultati intermedi.
Ogni scheda operazioni specifica (ed attiva) uno fra quattro dispositivi aritmetici, corrispondenti alle 4 operazioni. Le schede variabili specificano le locazioni di memoria da usare per una specifica operazione, cio i registri sorgente contenenti gli operandi, ed il registro destinazione su cui memorizzare il risultato.

Riportiamo di seguito un esempio di programma utilizzato per calcolare una risolvente di un sistema di equazioni di primo grado:

Carte Operazioni	**Programma Carte Sorgente**	**Variabili Destinazione**	**Commento**
X (moltipl.)	W2W4	W8	W8←a22b1
X	W1,W5	W9	W9←a12b2
X	W0,W4	W10	W10←a11a22
X	W1,W3	W11	.....
_ (sottr.)	W8,W9	W12	
_	W10,W11	W13	
/ (div.)	W12,W13	W14	W14 ← W12/W13

Babbage continu a progettare nuove versioni delle sue macchine, senza completare la realizzazione di nessuna. Il governo sospese gli aiuti ma, fino alla morte, Babbage continu nei suoi studi.
Il problema principale era l'inadeguatezza della tecnologia a disposizione: troppo lenta, troppo inaffidabile, troppo complessa.

1.2.2 L'era elettronica: ENIAC, EDVAC

Dopo le macchine di Babbage si dovette attendere fino al 1930 quando Konrad Zuse in Germania e Howard Aiken negli Stati Uniti costruirono due macchine molto simili alla Macchina Analitica di Babbage, utilizzando tecnologia elettromeccanica.
Nella macchina di Aiken, nota come Harvard Mark I, la memoria aveva una capacit di 72 numeri decimali di 23 cifre ciascuno. Le schede perforate erano di un solo tipo, ed avevano il formato:
A1 A2 OP

con il significato: A1 OP A2 $\longrightarrow$ A2
dove OP l'operazione da effetuare, A1 ed A2 sono i registri contenenti i dati su cui operare (*registri operandi*) . Il risultato veniva poi memorizzato su A2.

I calcolatori meccanici soffrivano di due limitazioni:
- la velocit di calcolo era limitata dall'inerzia delle parti mobili
- la trasmissione dell'informazione meccanica (ingranaggi, leve, ecc.) lenta, complessa, e poco affidabile.

Il primo elaboratore elettronico largamente conosciuto ENIAC, costruito nell'Universit della Pennsylvania sotto la direzione di John Mauchly (1907-1980) con lo scopo di automatizzare il calcolo delle tavole balistiche per l'esercito americano.
Questo elaboratore pesava 30 tonnellate, e conteneva oltre 18.000 valvole. Tuttavia, grazie all'uso della tecnologia elettronica, ENIAC era molto pi veloce del suo predecessore Mark I. Una moltiplicazione a 10 cifre richiedeva 3 ms anzich 3 secondi!!
Un ulteriore progresso fu registrato con EDVAC, alcuni anni dopo. Secondo un'idea del logico ungherese VonNeumann (di cui parleremo in seguito) EDVAC registrava in una stessa memoria programmi e dati, e dava la possibilit di modificare le istruzioni durante l'esecuzione di un programma.
Questa caratteristica, che ha parecchi inconvenienti ed ora scarsamente usata, rendeva per possibile scrivere in modo pi compatto programmi che eseguissero operazioni identiche su sequenze di dati memorizzati in locazioni adiacenti - una sorta di primitiva istruzione "*do for*..". Inoltre, EDVAC disponeva di un meccanismo di memorizzazione - le cos dette "linee di

ritardo", costituite da tubi di mercurio alle cui estremit sono applicati cristalli di quarzo - pi veloce e meno ingombrante delle valvole. Un'altra importante differenza consisteva nell'utilizzo del codice binario per la rappresentazione dell'informazione.

1.2.3 Von Neumann e la prima generazione

EDVAC fu profondamente influenzato dalle idee di VonNeumann, ma il primo calcolatore costruito interamente sotto la sua direzione, in collaborazione con Herman Goldstine, fu IAS (il calcolatore dell' Institute for Advanced Studies), noto come prototipo di calcolatore della prima generazione, o macchina di Von Neumann.
VonNeumann era un grande matematico ed esperto di logica. Per questa sua competenza, fu il primo a capire come fosse pi importante definire le *funzioni logiche* di un computer, piuttosto che concentrarsi sugli aspetti elettromeccanici, cio sull hardware.
Il suo contributo si basa su due intuizioni fondamentali:

separare le funzioni di controllo dell'elaborazione dalle funzioni di esecuzione

prevedere moduli hardware specifici per ognuna delle funzioni logiche fondamentali, ovvero:
controllo
esecuzione
memorizzazione
comunicazione
Nelle sue funzionalit generali, IAS ricalcava lo schema e le modalit di funzionamento della figura 1.1.

1.2.4 Le cinque generazioni

1a generazione. IAS, e le macchine costruite nello stesso periodo, note come calcolatori della prima generazione, avevano alcuni importanti limiti.
Dal punto di vista dell'hardware, la scarsa capacit di memorizzazione e la bassa velocit erano un forte limite alla complessit dei problemi risolvibili.
Dal punto di vista software, esistevano molte limitazioni, che rendevano il processo di programmazione complesso e tedioso:
- non esistevano linguaggi di "alto livello", ed occorreva programmare in codice binario
- non si potevano utilizzare variabili
- mancava il concetto di "ciclo" e gli strumenti software per supportarlo
- mancava qualsiasi tipo di supporto alla programmazione ed alla strutturazione dei programmi (sistemi operativi, librerie,..)
- l'unit di calcolo era poco orientata alla soluzione di problemi non numerici

2a generazione. La seconda generazione si fa coincidere con l'avvento dei transistors, nel 1948. Le memorie a linee di ritardo vengono sostituite con anelli di ferrite.
Dal punto di vista software, vengono introdotti meccanismi per realizzare cicli e definire variabili.
Vengono introdotti i primi linguaggi di alto livello: COBOL, FORTRAN, ALGOL, ed i primi supporti alla programmazione: librerie, subroutines, compilatori, monitors.
Le funzioni di comunicazione vengono assegnate ad elaboratori specializzati (I/O processors).

3a generazione. Si fa coincidere con l'avvento dei circuiti integrati, nel 1965. Le memorie vengono realizzate con tecnologia a semiconduttore. Dal punto di vista architetturale, viene introdotto il concetto di parallelismo: a vari livelli (di istruzione, di programma, ecc.) possibile velocizzare l'esecuzione di una certa funzione eseguendo pi passi in parallelo su moduli diversi.

Viene introdotto il concetto di microprogrammazione per semplificare il progetto dell'Unit di controllo.
Vengono introdotti i primi Sistemi Operativi, o software "di base" per la gestione della condivisione da parte di pi utenti delle risorse di un computer, ovvero spazio di memoria, processori, ecc.

4a generazione. Coincide con l'era dei VLSI, Very Large Scale Integration. Pur mantenendo le funzionalit logiche di base di una macchina di VonNeumann, l'avvento dei circuiti ad elevata integrazione consente di potenziare i concetti di parallelismo, di aumentare le funzionalit di elaborazione e controllo, nonch la capacit di memorizzazione.
Vengono realizzati i primi microprocessori, vengono realizzati elaboratori paralleli o supercomputers, vengono introdotte interfacce amichevoli per facilitare l'uso del computer da parte di non esperti di informatica.
Dal punto di vista software, vengono introdotti linguaggi orientati al parallelismo ed al *problem-solving.*

5a generazione. Si cerca di superare alcuni limiti inerenti la macchina di VonNeumann, per esempio la necessit di eseguire le istruzioni di un programma nell'ordine in cui sono state memorizzate.
In generale, le macchine della quinta generazione si propongono di affrontare problemi tipici dell'intelligenza umana, come la comprensione di immagini e del linguaggio naturale.
Le architetture cos dette connessioniste (controparte hardware dei sistemi neurali) prendono ispirazione da alcuni fatti noti sul funzionamento del cervello umano, ovvero:
- un grande numero di elaboratori molto semplici - in analogia con i neuroni
- un grande numero di connessioni "pesate" (ovvero con diversa intensit del legame): la "conoscenza" nella rete di connessione, piuttosto che nei singoli elementi di elaborazione;
- un controllo altamente parallelo
- capacit di "apprendere" automaticamente modificando lo stato della rete nel corso dell'elaborazione.

In generale, le architetture della 5a generazione puntano sulla molteplicit e versatilit delle comunicazioni fra elementi di elaborazione.

1.3 Rappresentazione dell'Informazione

I calcolatori elettronici sono macchine in grado di elaborare informazioni trasformandole in altre informazioni. Nel mondo dell'informatica, intendiamo in modo pi restrittivo per informazione *tutto ci che pu essere rappresentato tramite opportune sequenze di simboli in un alfabeto prefissato*.

La *rappresentazione* di un insieme I un insieme di parole ognuna delle quali esprime un elemento di I.
Esempio: {mela,pera,uva,arancia}
Un *codice* C un insieme di parole composte da simboli di un alfabeto Σ (detto *alfabeto di supporto* di C).
La *codifica* di un insieme di informazioni I in un dato codice C una funzione
f: I $\rightarrow$ C
Esempio:

mela $\longrightarrow$ 00, pera $\longrightarrow$ 01, uva $\longrightarrow$ 10, arancia $\longrightarrow$ 11, dove Σ = {0,1}

La *decodifica* di una informazione codificata in precedenza una corrispondenza g : C $\rightarrow$ I

Esempio:

Il cifrario di Cesare, usato nei tempi dell'antica Roma, aveva la seguente funzione di codifica:

f: i $\longrightarrow$ i+3(mod 26) i=0,1..25
dove al numero 0 corrisponde la lettera a, 1 a b ecc.
Secondo tale codice, la parola "babbo" codificata come "edeer" ecc;

Le diverse codifiche possono essere caratterizzate in base ad un insieme di propriet :
Economicit : sono considerate migliori rispetto a questa caratteristica le codifiche che utilizzano pochi simboli.
Semplicit di codifica e decodifica: auspicabile poter trasformare un linguaggio da un codice all'altro in modo efficiente
Semplicit di elaborazione: sono preferibili le codifiche che consentono di eseguire le operazioni definite sui dati in modo agevole (ad esempio, sostituendo ai simboli arabi i simboli dei numeri romani, "saltano" il meccanismo del riporto e della posizionalit).

1.3.1 Sistemi posizionali

Un sistema numerico *posizionale* in base *b* , ovvero basato su un alfabeto Σ di b simboli distinti, consente di esprimere un qualsiasi numero naturale N di m cifre, mediante la:

$$N = \sum_{i=0}^{m-1} c_i b^i, \quad c_i \in \Sigma$$

Ad esempio, nel sistema decimale (b=10, Σ=0,1,..9), la sequenza N^{10}= 284 può essere espressa come:

$$2x10^2+8x10^1+4x10^0$$

Tabella 2.1 Rappresentazioni dei numeri interi da 0 a 15 per diverse basi.

	Base 2	Base 3	Base 4	Base 5	…	Base 8	…	Base 10	…	Base 16
	0000	000	00	00		00		00		0
	0001	001	01	01		01		01		1
	0010	002	02	02		02		02		2
	0011	010	03	03		03		03		3
	0100	011	10	04		04		04		4
	0101	012	11	10		05		05		5
	0110	020	12	11		06		06		6
N_b	0111	021	13	12		07		07		7
	1000	022	20	13		10		08		8
	1001	100	21	14		11		09		9
	1010	101	22	20		12		10		A
	1011	102	23	21		13		11		B
	1100	110	30	22		14		12		C
	1101	111	31	23		15		13		D
	1110	112	32	24		16		14		E
	1111	120	33	30		17		15		F

1.3.2 Il sistema binario

Gli elaboratori digitali utilizzano, per rappresentare le informazioni al proprio interno, il codice binario costituito dai simboli 0 ed 1. I simboli 0 ed 1 prendono il nome di *bit*, una contrazione per "binary digit".
Il motivo per cui il sistema binario ha avuto tanta importanza nei sistemi di elaborazione dovuto al contributo di George Boole, il quale dimostr come la logica possa essere ridotta ad un sistema algebrico molto semplice, che utilizza solo un codice binario (zero e uno, vero e falso). Il codice binario fu trovato particolarmente utile nella teoria della commutazione per descrivere il comportamento dei circuiti digitali (1=acceso, 0=spento).
Claude Shannon defin un metodo per rappresentare un qualsiasi circuito costituto da un combinazione di interruttori (e/o r lais) mediante un insieme di espressioni matematiche, basate sulle regole dell'algebra Booleana.

Una cifra booleana di m bit pu essere rappresentata mediante la:

$$N = \sum_{i=0..m-1} c_i 2^i,\ c_i \in \{0,1\}$$

Il bit c_0 , viene detto LSB (less signifying bit) mentre c_{m-1} viene detto bit pi significativo , o MSB.

1.3.3 Cambiamenti di base e artimetica in base b

A1) RAPPRESENTAZIONE DEI NATURALI (interi positivi)

Sistema posizionale (in base b ≥ 2)

$c_{m-i}\, c_{m-2} \ldots c_1\, c_0 \quad = \quad \Sigma_{i = 0,\ldots,m-1}\, c_i\, b^i$

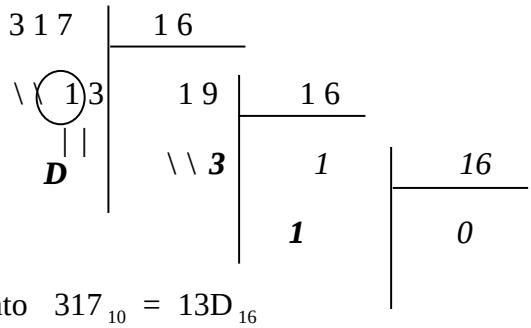
con $c_i \in \{ 0 , \ldots , b\text{-}1 \}$

Problema : passare da N_a a N_b (con N un naturale ed a e b le basi desiderate)

Algoritmo : 1. dividi N_a ripetutamente per b_a finchè il quoziente non è 0

2. i resti di queste divisioni (convertite da base a a base b) danno le cifre dalla meno alla più significativa di N_b

Esempio 1 : esprimere in base 16 il numero 317_{10}



Pertanto $317_{10} = 13D_{16}$

NOTATE che, nell'algoritmo sopra descritto, la divisione **va eseguita in base a (cioè nella base del numero di partenza)**. Se a≠10, questo può risultare complicato.

Esempio 2 : convertire il numero 102202 da base 3 a base 5

Due strade :

a) eseguire $102202 /_3 12$ (cioè effettuare la divisione in base 3)
=> DIFFICILE

b) convertire 102202 in base 10 e poi convertire il risultato in base 5
- $102202_3 = 3^5 + 2{\cdot}3^3 + 2{\cdot}3^2 + 2 = 317_{10}$
- col procedimento dell'esempio precedente $317_{10} = 2232_5$

Prop. : *lavorando il aritmetica in base b si ha che*
1. $n_{m-1} \ldots n_1\, n_0 \;\; DIV \;\; b^i = n_{m-1} \ldots n_i \;\; r = n_{i-1} \ldots n_0$
2. $n_{m-1} \ldots n_1\, n_0 \;\; MOD \;\; b^i = n_{i-1} \ldots n_0$

(r = resto, MOD = modulo)

Es: 353_{10} DIV 10^0 =35 (r=3) 1011_2 DIV 10_2 (cioè 2^1) = 101 r=1

Da ciò :

Conversione da base 2 a base 4 : considera i bit a coppie partendo dal meno significativo e traducile in base 4

Esempio 3 : convertire 100111101 da base 2 a base $4=2^2$

$$1\ 00\ 11\ 11\ 01_2 = 1\ 0\ 3\ 3\ 1_4$$

(sequenza dei resti per successive divisioni per 2^2.

Conversione da base 2 a base 8 : considera i bit a triple partendo dal meno significativo e traducile in base 8

Esempio 4 : convertire 100111101 da base 2 a base $8=2^3$

$$100\ 111\ 101_2 = 4\ 7\ 5_8$$

Conversione da base 2 a base $16=2^4$: considera i bit a quadruple partendo dal meno significativo e traducile in base 16

Esempio 5 : convertire 100111101 da base 2 a base 16

$$1\ 0011\ 1101_2 = 1\ 3\ D_{16}$$

Vale anche il viceversa. In particolare :

Conversione da base 8 a base 2 : considera ogni cifra ottale e riscrivila come numero binario di 3 bit

Conversione da base 16 a base 2 : considera ogni cifra esadecimale e riscrivila come numero binario di 4 bit

A2. ARITMETICA IN BASE b PER I NATURALI

Tutte le operazioni vengono eseguite come in base 10 , ma *modulo b* (Es.: $(1+1)_2 = 10_2$)
e quindi anche i riporti e i prestiti agiscono *modulo b*. In base 2 si ha:
0+0=0, riporto=0
0+1=1+0=1. rip=0
1+1=0, rip=1

Nota: non vi ricordate cos'è il modulo di un numero? http://www.math.csusb.edu/notes/rel/node4.html
In generale, usate http://www.math.csusb.edu/notes/ quando avete dei dubbi, per ottenere definizioni di concetti matematici che non ricordate o che non possedete.

Esempio 6 : sommare in base 2 i numeri 1001 e 111

$$\begin{array}{r} 1\ 0\ 0\ 1 \ + \\ 1\ 1\ 1 \ = \\ \hline 1\ 0\ 0\ 0\ 0 \end{array}$$

Nota: Un elaboratore lavora con parole di lunghezza fissa (diciamo *n*); quindi
se un numero è codificato con meno di *n* bit dobbiamo inserire in testa (*n-m*) zeri non significativi

se un numero è codificato con più di *n* bit : dobbiamo considerare solo le *n* cifre meno significative (situazione di errore detta *overflow* e gestita con un'eccezione del processore)

Esempio 6 (continua) : usando parole lunghe quattro bit avremmo

```
  1 0 0 1  +
    1 1 1  =
-------------
  0 0 0 0
```

che è chiaramente un errore!! Il risultato infatti è 10000, non rappresentabile con 4 bit.

B RAPPRESENTAZIONE DEGLI INTERI

Rispetto ai naturali, il problema è la rappresentazione del segno. Esistono tre modalità di rappresentazione: in **modulo e segno**, in **complemento a uno** e in **complemento a due**. I primi due rendono le operazioni di somma e sottrazione delicate (sono necessari controlli preliminari sul segno e sui valori assoluti degli operandi); col secondo, invece, la sottrazione si esegue semplicemente come somma dell'opposto (a patto di ignorare l'eventuale overflow derivante dalla somma di numeri negativi).

Tabella 2.6 Rappresentazioni binarie su 4 bit di numeri interi con segno.

	Valore rappresentato		
Valore binario $b_3b_2b_1b_0$	**Notazione modulo e segno**	**Notazione complemento a 2**	**Notazione complemento a 1**
0000	+0	0	0
0001	+1	+1	+1
0010	+2	+2	+2
0011	+3	+3	+3
0100	+4	+4	+4
0101	+5	+5	+5
0110	+6	+6	+6
0111	+7	+7	+7
1000	−0	−8	−7
1001	−1	−7	−6
1010	−2	−6	−5
1011	−3	−5	−4
1100	−4	−4	−3
1101	−5	−3	−2
1110	−6	−2	−1
1111	−7	−1	−0

B.1) RAPPRESENTAZIONE IN COMPLEMENTO A DUE

In complemento a due (Ca2) si ha che: $A = -c_{m-1} \times b^{m-1} + \sum_{i=0}^{m-2} c_i \times b^i$ (b= base, come al solito)

$$\overline{A_{Ca2}} = \overline{c_{m-1}}\ \overline{c_{m-2}} \ldots \overline{c_0} + 1$$

dove $\overline{c_i} = 1 - c_i$.

Esempio per m=3

011 ⟶ +3
010 ⟶ +2
001 ⟶ +1
000 ⟶ 0
100 ⟶ -4
101 ⟶ -3
110 ⟶ -2
111 ⟶ -1

Osservazioni:

il "range" dei numeri rappresentabili va da $-b^{m-1}$ *a* $b^{m-1} - 1$ (da –4 a +3 nll'esempio precedente)

lo zero è $\underbrace{0 \ldots 0}_{m}$

non viene considerata la sequenza $1\underbrace{0 \ldots 0}_{m-1}$

il bit più significativo NON E' un bit di segno (nel senso che per complementare un numero non basta complementarne il bit più significativo) ma è un bit indicatore del segno; infatti

$$c_{m-1} = \begin{cases} 1 & \text{se } c < 0 \\ 0 & \text{altrimenti} \end{cases}$$

NOTA: Negli esempi seguenti supporremo sempre di lavorare **con parole di 8 bit.**

Esempio 7 : complementare i seguenti numeri binari : 10010 e 11011000

a) 1. complementa bit a bit il numero (00010010 diventa 11101101)
2. somma 1 (11101101 diventa 11101110)

Nota: Gli 1 e gli 0 aggiunti servono per ottenere parole da 8 bit, e in Ca2 NON variano il valore del numero rappresentato.

b) 1. complementa bit a bit il numero (11011000 diventa 00100111)
2. somma 1 (00100111 diventa 00101000)

Mostriamo ora come le operazioni aritmetiche in questo caso siano molto semplici.

Esempio 8 : eseguire in base 2 le seguenti operazioni espresse in base 10 :
a) 6 + 8 b) – 6 + 8 c) 6 – 8 d) – 6 – 8

Anzitutto traduciamo 6 , – 6 , 8 e – 8 in base 2
. $6_{10} = 00000110_2$
. $-6_{10} = 11111010_2$
. $8_{10} = 0001000_2$
. $-8_{10} = 11111000_2$
omettiamo gli zeri più significativi per brevità.

a) 110 + 1000 = 1110 (cioè 14_{10})
b) 11111010 + 1000 = 10 (cioè 2_{10})
c) 110 + 11111000 = 11111110 (cioè -2_{10})
d) 11111010 + 11111000 = 11110010 (cioè -14_{10})

C) RAPPRESENTAZIONE DEI REALI

Il problema aggiuntivo è la rappresentazione della parte intera e di quella frazionaria.

Abbiamo sempre un sistema posizionale (in base b ≥ 2). I primi m bit rappresentano la parte intera, i successivi n la parte frazionaria.

$$\underbrace{c_{m-1}\cdots c_0}_{\text{parte intera}}\ \underbrace{c_{-1}c_{-2}\cdots c_{-n}}_{\text{parte frazionaria}} = \sum_{i=m-1}^{-n} c_i b^i$$

con $c_i \in \{ 0 , \dots , b\text{-}1 \}$

C.1) VIRGOLA FISSA

Riserva m bit per la parte intera (P.I.) e n bit per la parte frazionaria (P.F.) (m e n fissati)

Cambiamento di base : trasformare $<A , B_a>$ in $<A , B_b>$

A indica la PI, B indica la PF.

1. converti A_a in A_b (normale conversione per naturali)

moltiplica B_a per b_a

- la P.I. del risultato convertita in base b è la 1^ cifra di B_b
- itera il procedimento sulla P.F. del risultato
- finché la P.F. è 0, **oppure** sono state determinate tutte le cifre disponibili per esprimere la P.F. di B_b

Esempio 1 : convertire 17,416 in base 2 con 8 bit sia per P.I. che per P.F.

1. $17_{10} = 10001_2$
2. - 0,416 * 2 = 0,832 da cui P.I. = 0 P.F. = 0,832
   - 0,832 * 2 = 1,664 da cui P.I. = 1 P.F. = 0,664
   - 0,664 * 2 = 1,328 da cui P.I. = 1 P.F. = 0,328
   - 0,328 * 2 = 0,656 da cui P.I. = 0 P.F. = 0,656
   - 0,656 * 2 = 1,312 da cui P.I. = 1 P.F. = 0,312
   - 0,312 * 2 = 0,624 da cui P.I. = 0 P.F. = 0,624
   - 0,624 * 2 = 1,248 da cui P.I. = 1 P.F. = 0,248
   - 0,248 * 2 = 0,496 da cui P.I. = 0 P.F. = 0,496

Perciò $0{,}416_{10} = 0{,}01101010_2$

Pertanto $17{,}416_{10} = 00010001\ ,\ 01101010_2$

N.B.: la versione binaria è un'approssimazione del numero decimale originale.
Infatti :

$10001{,}0110101_2 = 2^4 + 1 + 2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} = 17{,}4140625_{10}$

Problema: l'intervallo dei reali rappresentabile è piccolo e con approssimazioni grossolane

Esempio 2 : avendo a disposizione 32 bit e dandone 20 per la P.I. e 12 per la P.F. si ha
a) P.I. $\in \{ 2^{-19} + 1 , \dots , 2^{19} - 1 \} = \{ -524287 , \dots , 524287 \}$
b) si hanno a disposizione solo 3 o 4 cifre frazionarie (infatti $2^{12} = 4096$)

C.2) VIRGOLA MOBILE

Un reale r è rappresentato dalla terna $< s , m , e >$ dove

$r = (-1)^s \cdot m \cdot b^e$

e gli elementi della terna sono chiamati rispettivamente s= *bit di segno* , m= *mantissa* (o *significante*) e e = *esponente* , espresso in Ca2.
Tipicamente si adotta una forma normalizzata (tranne che per lo zero) in cui la mantissa è tale che :
1. la sua parte intera è nulla
2. la sua parte frazionaria inizia con una cifra non nulla

Banalmente $< s , m , e >$ soddisfa ciò se e solo se $b^{-1} \le m < 1$.

Quindi, adottando la rappresentazione normalizzata,

$r = (-1)^s \cdot 0{,}m \cdot b^e$

dove
s è il bit di segno della mantissa
m (m è un intero) rappresenta la parte frazionaria del numero normalizzato (quindi la mantissa è un intero rappresentato con bit di segno)
e è l'esponente, rappresentato in complemento alla base (se b=2, in complemento a 2)

C.2.1) CAMBIAMENTO di BASE

Trasformare $< s , m_a , e_a >$ in $< s , m_b , e_b >$

1. applica il procedimento per numeri in virgola fissa a $(0{,}m_a \cdot a_e)_a$ ottenendo h , k_b
2. m_b e e_b sono la mantissa e l'esponente normalizzati di h , k_b

Nel seguito assumeremo di avere 1 bit per il segno, 8 per la mantissa e 4 per l'esponente.

Esempio 3 : convertire in base 2 il numero $0{,}09375_{10}$ = <+,0,9375,-1>

1. applico il procedimento per la P.F., ottenendo
$0{,}09375_{10} = 0{,}00011_2 = 0{,}11 \times 2^{-3}$
2. la rappresentazione cercata è $< 0 , 11000000_2 , 1101_2 >$
(-3 in complemento a 2 è appunto 1101)

Esempio 4 : convertire in decimale i seguenti numeri in virgola mobile :

a) $< 0 , 10010000_2 , 0101_2 > = + (2^{-1} + 2^{-4}) \cdot 2^{5} = 18_{10}$
b) $< 1 , 11001000_2 , 0111_2 > = - (2^{-1} + 2^{-2} + 2^{-5}) \cdot 2^{7} = -100_{10}$
c) $< 1 , 10001000_2 , 1101_2 > = - (2^{-1} + 2^{-5}) \cdot 2^{-3} = -0{,}06640625_{10}$

N.B. : in base 2, intervallo rappresentato dando M bit alla mantissa e E all'esponente :

1. i numeri positivi sono $[\, 0{,}1 \times 2^{-2^{E-1}} , 0{,}1\underbrace{.....}_{M}1 \times 2^{2^{E-1}-1} \,]$
2. i numeri negativi sono $[\, -0{,}1\underbrace{.....}_{M}1 \times 2^{2^{E-1}-1} , -0{,}1 \times 2^{E-1} \,]$

Tabella 2.8 Relazione tra numero di bit di mantissa ed esponente ($n_m = 4$ e $n_e = 3$).

	$M = 0{,}5$	$M = 0{,}625$	$M = 0{,}75$	$M = 0{,}875$
$E = -4$	0,03125	0,0390625	0,046875	0,0546875
$E = -3$	0,0625	0,078125	0,09375	0,109375
$E = -2$	0,125	0,15625	0,1875	0,21875
$E = -1$	0,25	0,3125	0,375	0,4375
$E = 0$	0,5	0,625	0,75	0,875
$E = 1$	1	1,25	1,5	1,75
$E = 2$	2	2,5	3	3,5
$E = 3$	4	5	6	7

Tabella 2.9 Relazione tra numero di bit di mantissa ed esponente ($n_m = 5$ e $n_e = 2$).

	M=0,5	M=0,5625	M=0,625	M=0,6875	M=0,75	M=0,8125	M=0,875	M=0,9375
E=−2	0,125	0,140625	0,15625	0,171875	0,1875	0,203125	0,21875	0,234375
E=−1	0,25	0,28125	0,3125	0,34375	0,375	0,40625	0,4375	0,46875
E=0	0,5	0,5625	0,625	0,6875	0,75	0,8125	0,875	0,9375
E=1	1	1,125	1,25	1,375	1,5	1,625	1,75	1,875

Come si vede, all'aumentare del numero di bit della mantissa aumenta la precisione della rappresentazione (diminuiscono gli intervalli fra numeri adiacenti), mentre all'aumentare del numero di bit dell'esponente aumenta l'ampiezza del campo dei numeri rappresentabili.
Occorre dunque trovare un compromesso.

Per uniformare la gestione della rappresentazione in virgola mobile nei vari sistemi digitali (ad evitare cioè che gli stessi dati elaborati su sistemi digitali diversi diano risultati diversi) l'IEEE ha emesso degli standard di rappresentazione.

Tabella 2.11 I quattro formati standard IEEE 754 - 1985.

	Precisione			
	Singola	**Singola estesa**	**Doppia**	**Doppia estesa**
***P* (bit della mantissa)**	23	≥ 31	52	≥ 63
e_{max}	127	≥ 1023	1023	≥ 16383
e_{min}	– 126	≤ – 1022	– 1022	≤ – 16382
Bias	127	Non specificato	1023	Non specificato
Larghezza esponente	8	≥ 11	11	≥ 15
Larghezza formato	32	≥ 43	64	≥ 79

La rappresentazione utilizza una polarizzazione, o bias, cioè un valore costante che viene sommato all'esponente *e*, per ottenere un esponente *polarizzato*.
La figura 2.12 mostra degli esempi. NaN è lo standard di rappresentazione di Not a Number. Inoltre vengono mostrate le convenzioni di rappresentazione per 0 e per infinito.

Tabella 2.12 Esempi di interpretazione di numeri in virgola mobile nel formato IEEE 754 - 1985 in singola precisione.

s	E	m		
0	0000 0000	0000 0000 0000 0000 0000 000	=	+ 0
1	0000 0000	0000 0000 0000 0000 0000 000	=	− 0
0	1111 1111	0000 0000 0000 0000 0000 000	=	+ ∞
1	1111 1111	0000 0000 0000 0000 0000 000	=	− ∞
0	1111 1111	0000 0100 0000 0000 0000 000	=	*NaN*
1	1111 1111	0010 0010 0010 0101 0101 010	=	*NaN*
0	1000 0000	0000 0000 0000 0000 0000 000	=	$+1 \times 2^{(128-127)} \times 1{,}0 = 2$
0	1000 0001	1010 0000 0000 0000 0000 000	=	$+1 \times 2^{(129-127)} \times 1{,}101 = 6{,}5$
1	1000 0001	1010 0000 0000 0000 0000 000	=	$-1 \times 2^{(129-127)} \times 1{,}101 = -6{,}5$
0	0000 0001	0000 0000 0000 0000 0000 000	=	$+1 \times 2^{(1-127)} \times 1{,}0 = 2^{(-126)}$
0	0000 0000	1000 0000 0000 0000 0000 000	=	$+1 \times 2^{(-126)} \times 0{,}1 = 2^{(-127)}$
0	0000 0000	0000 0000 0000 0000 0000 001	=	$+1 \times 2^{(-126)}\ 0{,}00000000000000000000001$
				$= 2^{(-149)}$(più piccolo valore positivo)

C.2.2) OPERAZIONI tra REALI

Moltiplicazione (in base b)

$< s_1 , m_1 , e_1 > * < s_2 , m_2 , e_2 > = < s , m , e >$

dove 1. $s = \begin{cases} 0 & \text{se } s_1 = s_2 \\ 1 & \text{altrimenti} \end{cases}$

2. m ed e sono la mantissa e l'esponente normalizzati di $m_1 \cdot m_2 \cdot b^{e1+e2}$

N.B. : attenzione all'overflow degli esponenti!!

Analoga è la formula per la divisione: $m_1 \div m_2 . b^{e1-e2}$.

Esempio 5 : eseguire in base 2 18 * -0,06640625
< 0 , 10010000 , 0101 > * < 1 , 10001000 , 1101 > = < 1 , 10011001 , 0001 >
Il risultato, convertito in base 10, è correttamente – 1,1953125

Somma

$< s_1 , m_1 , e_1 > + < s_2 , m_2 , e_2 > \; = \; < s , m , e >$

- **se** $e_1 = e_2$

  - $s = \begin{cases} s_1 & \text{se } (-1)^{s1} \cdot m_1 \geq (-1)^{s2} \cdot m_2 \\ s_2 & \text{altrimenti} \end{cases}$

  - m ed e sono le normalizzazioni di m' ed e' definiti come :

  (i) $e = e_1$

  (ii) $m = \begin{cases} m_1 + m_2 & \text{se } s_1 = s_2 \\ | m_1 - m_2 | & \text{altrimenti} \end{cases}$

Es: $-0{,}3\times10^{-2} + 0{,}4\times10^{-2} = +0{,}1\times10^{-2}$

- **altrimenti** (sia $e_1 < e_2$)
  - shift a destra di m_1 di $e_2 - e_1$ posizioni (inserendo 0 a sinistra)
- porta così i numeri allo stesso esponente
- procede come al punto precedente

Esempio 6 : eseguire in base 2 18 – 100

$< 0 , 10010000 , 0101 > - < 1 , 11001000 , 0111 >$
=(e1=+5, e2=+8, $e_2 - e_1$=2)

$= \; < 0 , \mathbf{00}100100 , 0111 > - < 1 , 11001000 , 0111 > \; =$

$= \; < 0 , 10100100 , 0111 >$

che corrisponde a – 82

N.B. : nell'operazione di shift ci può essere perdita di cifre significative !!

Esempio 7 : si esegua in base 2 18 – 0,06640625

$< 0 , 10010000 , 0101 > - < 1 , 10001000 , 1101 > \; =$
$= \; < 0 , 10010000 , 0101 > - < 1 , 00000000 , 0101 > \; =$
$= \; < 0 , 10010000 , 0101 >$

Nello shift si è addirittura perso il secondo operando !!!!

ESERCIZI DA SVOLGERE

Convertire il numero decimale 1342 nelle basi 2 , 3 , 5 , 6 , 7 , 9 , 16

Convertire il numero binario 1011100 nelle basi 3 , 4 , 5 , 8 , 10 , 16

Dimostrare la proposizione data (*Sugg. : usare il teorema fondamentale dell'algebra*)

Convertire in base 2 i seguenti numeri ottali : 742 , 1176 , 253 , 1064

Convertire in base 2 i seguenti numeri esadecimali : A231 , 1BC , 1045 , FA2

Convertire in base 2 e poi sommare le seguenti coppie di naturali espressi in base 10. Quali di queste somme danno problemi di overflow rappresentando i numeri con 8 bit (= 1 byte) ?
a) 115 , 64 b) 83 , 12 c) 197 , 94 d) 241 , 16 e) 230 , 25 f) 107 , 44

Rappresentare in complemento a due gli opposti dei seguenti interi e convertirli in base 10 :
a) 1001 b) 11110010 c) 11010010 d) 10010
e) 111000 f) 11001011 g) 1000111 h) 11101110

Eseguire in base 2 con numeri rappresentati come byte le seguenti operazioni decimali :
a) 16 – 4 b) 18 – 47 c) – 49 + 54 d) – 4 –101 e) 92 + 23

Convertire in binario i seguenti numeri razionali, usando 8 bit sia per la P.I. Che per la P.F.
a) 27,311 b) 8,92 c) 0,511 d) 107,88 e) 49,266

Convertire in decimale i seguenti numeri razionali binari espressi con le convenzioni dell'esercizio precedente
a) 11001,11100011 b) 100011,00100111 c) 110,00010111 d) 111011,00001111

Convertire in binario i seguenti numeri razionali, usando 1 bit di segno, 8 bit per la mantissa e 4 bit per l'esponente
a) 27,311 b) - 8,92 c) 0,511 d) – 107,88
e) 49,266 f) 0,000615 g) – 0,00215 h) 1500,615

Convertire in decimale i seguenti numeri razionali binari espressi con le convenzioni dell'esercizio precedente
a) < 0 , 11100000 , 0111 > b) < 1 , 01100100 , 1110 >
c) < 1 , 10010000 , 0011 > d) < 0 , 01001000 , 0100 >

Moltiplicare e sommare tutte le possibili coppie di numeri dell'esercizio precedente (cioè a+b , a+c , a+d , b+c , b+d , c+d e analogo per *) segnalando eventuali problemi di overflow e di perdita di cifre significative.

1.4 Algebra Booleana

Il contributo di Boole fu di "dare espressione alle leggi fondamentali del ragionamento nel linguaggio simbolico del Calcolo".
Nella sua prima proposizione egli suppone di avere:

- un sistema di simboli arbitrari quali x,y, z.. i quali rappresentano gli oggetti che devono essere discussi;
- due operazioni designate, + e Æche operano sui simboli producendo ulteriori simboli giacch , se x ed y sono simboli, lo sono anche x+y e xÆy;
- un'operazione di identit =;
- Svariate regole che governano il comportamento delle operazioni, ad esempio:

x+y=y+x, xÆy=yÆx, xÆ(y+z)=xÆy+xÆz

Di fatto, queste regole sono le stesse dell'algebra classica, con una sola eccezione:

$x^2=xÆx=x$

Per capire la natura di questa - importantissima!- eccezione dobbiamo discutere la natura dei simboli ed il significato delle operazioni fra esse. Diciamo che, in generale, essi possono essere cose o gruppi o classi. Ad esempio, se x sta per "oggetti neri" e y per "mucche", allora xÆysta per "mucche nere" ovvero, una entit che ha entrambe le propriet x ed y.
Data questa spiegazione dei simboli e dell' operatore Æ, ovvio che xÆx=xperch l'intersezione di un insieme con s stesso ancora l'insieme di partenza, ovvero, xÆx¡, per analogia con la teoria degli insiemi, x∩x non esprime nulla di pi di x.
L'operazione + usata per porre in relazione entit disparate, e raggrupparle in una nuova entit o classe.
Se z rappresenta, ad esempio, la classe degli animali maschi, e k la classe degli animali femmina, z+k rappresenta l'unione delle due classi (cio z∪k), ovvero la classe degli animali maschi o femmine.
In questo modo, possibile rappresentare proposizioni complesse, ad esempio:

x·y·k+ x·z

rappresenta l'insieme delle mucche nere femmina e degli animali maschi neri. Ovviamente, y.k=y poich si suppone che le mucche siano tutte femmine!
Boole scelse di utilizzare il sistema binario come insieme di possibili valori per le sue variabili perch "le leggi, gli assiomi ed i procedimenti di un'algebra siffatta saranno identici in tutto e per tutto alle leggi, gli assiomi ed ai procedimenti di un'algebra della Logica. Li divideranno solo differenze di interpretazione".
La ragione di 0 ed 1 che , naturalmente, sono le sole quantit che soddisfano la xÆx=x!
Ma cosa significano questi simboli nella logica di Boole?
Consideriamo lo 0: in algebra, 0Æy=0. Boole afferma che "una semplice riflessione mostrer che questa condizione ha senso se 0 rappresenta il Nulla" Infatti, facile convincersi che nessuna propriet pu riferirsi ad un insieme di oggetti pi limitato di quello compreso in Nulla!
Il simbolo 1 caratterizzato in algebra dalla relazione1Æy=yper ogni y. Boole aggiunge che "una semplice riflessione mostrer che la classe qui rappresentata dal simbolo 1 deve essere l'Universo, giacch questa la sola classe in cui si trovano tutti gli oggetti che si trovano in una qualsiasi classe". (Boole: "Laws of Thought").
In questo corso non ci interessiamo di logica, ma dell'utilizzo dell'algebra di Boole per la rappresentazione del funzionamento dei circuiti digitali.
Da un punto di vista "fisico" i valori 0 e 1, nel limitato universo dei circuiti digitali, rappresentano i due valori della tensione in uscita ad un dispositivo elettronico che lavora in saturazione, ad esempio +5V e 0V, "alto" e "basso", "presenza"(di tensione) ed "assenza", o ancora "acceso" o "spento".

Come gi accennato in precedenza, Shannon svilupp un metodo di analisi di circuiti realizzati con relais basato sull'algebra di Boole. Poich i moderni circuiti digitali funzionano come relais (alto/basso, acceso/spento)), il metodo di Shannon applicabile ai moderni circuiti digitali.
Ci sono molti vantaggi legati alla possibilit di utilizzare tecniche matematiche per descrivere le operazioni interne di un dispositivo digitale.

¥ Primo, da un punto di vista progettuale, molto pi facile sintetizzare il funzionamento di un circuito manipolando delle espressioni, piuttosto che usare diagrammi a blocchi o schemi circuitali.

¥ Inoltre, queste espressioni possono essere semplificate applicando le regole dell'algebra ordinaria, ed i corrispondenti circuiti possono anch'essi essere semplificati.

¥ Infine, anche da un punto di vista descrittivo, l'algebra booleana consente di descrivere in modo semplice e compatto il funzionamento di un dispositivo digitale.

Pi formalmente:

L'algebra Booleana un'algebra definita su un supporto Σ di cardinalit due, ad esempio (0,1) (falso,vero) (nulla,universo), ecc.
Variabili booleane sono simboli a cui possono essere associati elementi di Σ.
Gli *operatori booleani* utilizzati vengono chiamati, nel linguaggio descrittivo del funzionamento dei circuiti digitali, AND (·) OR (+) e NOT (sopralineatura).
Simboli alternativi sono: $\wedge \otimes$ (AND) $\vee \oplus$ (OR) $\neg \sim$ (NOT). Ancora, queste operazioni vengono denominate rispettivamente, moltiplicazione logica, somma logica, e negazione.
Espressioni Booleane
Dato un insieme numerabile di variabili *Var* a valori binari, le espressioni booleane EB su di esso sono definite induttivamente da :

$0, 1 \in$ EB
$\forall\, v \in Var \quad v \in$ EB
se $E_1, E_2 \in$ EB allora $E_1 + E_2$, $E_1 \cdot E_2$, $\overline{E_1} \in$ EB

Assiomi dell'algebra di Boole:

Tabella 3.1 Identità e teoremi dell'algebra Booleana.

	AND	***OR***
Identità	$1 \cdot x = x$	$0 + x = x$
Elemento nullo	$0 \cdot x = 0$	$1 + x = 1$
Idempotenza	$x \cdot x = x$	$x + x = x$
Inverso	$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$
Commutativa	$x \cdot y = y \cdot x$	$x + y = y + x$
Associativa	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	$(x + y) + z = x + (y + z)$
Assorbimento	$x \cdot (x + y) = x$	$x + x \cdot y = x$
Distributiva	*AND* **rispetto a** *OR* $x \cdot (y + z) = x \cdot y + x \cdot z$	*OR* **rispetto a** *AND* $x + y \cdot z = (x + y) \cdot (x + z)$
De Morgan	$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$

Esistono molte leggi derivate da tali assiomi; molto importanti sono le leggi di **De Morgan** :

$$\overline{(x + y)} = \overline{x} \cdot \overline{y}$$

$$\overline{(x \cdot y)} = \overline{x} + \overline{y}$$

Grazie a questi teoremi, é possibile ricavare il complemento di un'espressione booleana o parte di essa. Per ottenere il complemento, in questi teoremi, si compiono due passi:
1. I simboli di somma vengono sostituiti dai simboli di moltiplicazione e viceversa.
2. Ognuno dei termini dell'espressione viene complementato.

I teoremi di De Morgan esprimono il *principio di dualità* dell'algebra di Boole. Ogni espressione booleana, ad esempio x+y+z , che è espressa in termini di operatori or, ha una sua duale, xyz, espressa in termini di operatori and, e vale la:

$$x + y + z = \overline{\overline{x + y + z}} = \overline{\overline{x} \cdot \overline{y} \cdot \overline{z}}.$$

Ed inoltre:

x(y + z)=xy + xz
x + xy=x
x(x + y)=x
(x + y)(x + z)=x + yz
x + xy=x + y
xy + yz + yz=xy + z

dimostrabili utilizzando gli assiomi della tyabella 3.1.