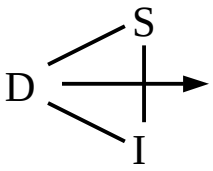


Fondamenti di Informatica
Parte 1

CCL - Amb-Ter

Facoltà di Ingegneria
Università degli Studi di Firenze

Ver.: 1.0
Data: 22/2/2000

Questo documento fa parte di una serie di dispense prodotte dallo sforzo volontario di molti ma che sono state riviste e corrette dal docente solo in parte. Anche se in questo stato preliminare ho deciso di distribuirle per dare comunque una traccia sugli argomenti del corso in mancanza di un libro di riferimento. Non rappresentano pertanto ne' il libro ne' il programma di riferimento del corso, a questo fine fanno fede le lezioni stesse che in molte parti differiranno da queste dispense.
Si prega pertanto di segnalare ogni mancanza e correzione inviando una mail al Prof. P. Nesi al seguente indirizzo di posta elettronica con soggetto "dispense": nesi@dsi.unifi.it , con la speranza di arrivare a produrre una versione rivista in breve tempo, anche con il Vostro aiuto, grazie!

INDI CE

1 Introduzione

1.1 Algoritmi

Il concetto di algoritmo e' uno dei concetti basilari della matematica. La parola algoritmo deriva dal nome del matematico arabo Al-Khuwarizmi vissuto nel IX secolo d.C.
Un algoritmo descrive un procedimento per risolvere una classe di *PROBLEMI in un numero finito di passi* descritti in modo rigoroso. La formalizzazione di un algoritmo significa trascrivere l'algorimo da una scrittura informale ad una prettamente formale ed avvolte anche matematica. Un algoritmo e' un elenco finito di istruzioni, di passi, che devono *essere* eseguiti per arrivare alla soluzione finale del problema. (e.g., per fare una torta c'è bisogno di una ricetta, che va seguita punto per punto). Gli algoritmi fanno uso di dati di ingresso e sono in grado di produrre dei risultati che costituiscono la soluzione del problema in questione. Se l'algoritmo non produce risultati serve a poco.

ESEMPIO DI ALGORITMO: "Calcolo delle radici di un polinomio di 2^0 grado"

Dato il polinomio $ax^2+bx+c=0$, si possono calcolare le radici x_1 e x_2 con il seguente algoritmo:

$$\Delta = b^2 - 4*a*c$$

$$x_{1,2} = (-b \pm \sqrt{\Delta}) / 2*a \qquad \text{supponendo il } \Delta >= 0 ;$$

Algoritmo dell'esempio e' stato formalizzato con una notazione a voi nota, la notazione matematica.

I dati costituiscono le informazioni che vengono fornite dall'esterno dall'utente -- e.g., a, b, ed c.
L'algoritmo acquisisce dei dati dall'esterno e comunica i risultati all'ambiente esterno. I risultati sono il prodotto dell'algoritmo, in questo caso le radici dell'equazione oppure la torta nel caso della ricetta.
L'algoritmo descrive come i risultati vengono prodotti in base ai dati. L'algoritmo e' composto di passi e di relazioni fra passi. L'algoritmo puo' essere eseguito da l'uomo o da macchine automatiche o da l'uomo con l'ausilio di macchine automatiche. In questo caso l'uomo esegue le istruzioni indicate nell'algoritmo sui dati che caratterizzano il particolare problema. L'algoritmo deve essere comprensibile per l'esecutore sia che questo sia un uomo che un macchina. Lo schema di esecuzione degli algoritmi e' costituito da alcune regole che indicano l'ordine di esecuzione delle istruzioni che lo costituiscono.

ESEMPI:
- Andare a fare la spesa
- Il calcolo di un prodotto
- Andare a iscriversi all'universita'
- Istruzioni per il noleggio
- Prelievo con bancomat
- Fare un torta
- Data una serie di numeri trovare il maggiore
- Dati due numeri trovare il MCD

- Dati due numeri trovare il mcm

Vediamo come puo' essere strutturato un algoritmo riguardante la ricetta per cucinare una torta:

"Battere in una ciotola 100 g di burro fin tanto che risultera' una cremina. Aggiungere poco per volta 150 g di farina, 2 uova, 1pizzico di sale, 100g di zucchero e 250 g di latte ottenendo una pasta piuttosto fluida.Versare l'impasto in una tortiera imburrata e mettere in forno a 180° per circa 1 ora".

L'insieme di operazioni da compiere e' rigorosamente ordinato e nel loro insieme determinano una serie di azioni concrete da svolgere secondo determinate regole. Il risultato finale, se tutti i passi sono stati eseguiti correttamente, e' la torta, altrimenti viene prodotto qualcosa di diverso…..

1.2 Proprieta' fondamentali degli algoritmi

Un algoritmo e' stato definito come un elenco di istruzioni in grado di risolvere una classe di problemi: ma in generale affinche' un elenco di istruzioni possa essere considerato un algoritmo debbono essere soddisfatti dei precisi requisiti. Per esempio tali requisiti posson essere:

Completezza: per ogni possibile combinazione delle variabili di ingresso e per ogni possibile sua sequenza l'algoritmo deve essere in grado di produrre una qualche uscita.
Ad esempio, l'algoritmo che riguarda la preparazione di una torta, perche' possa definirsi completo deve specificare la variazione del dosaggio degli ingredienti in proporzione al numero delle persone :

"Per 4 persone occorrono 100 g di farina; aggiungere altri 20 g per ogni persona in piu'. "

Consistenza: nella descrizione dell'algoritmo non vi sono punti in cui si deve fare un cosa ed il suo opposto, o comunque non si hanno conflitti nella produzione dei risultati.

Se A>0 allora fate … xyz……
Altrimenit sie A>3 allora fate …gyu……..

e' inconsistente

Rifacendosi sempre all'esempio della ricetta:

Se ho 4 persone: 10 grammi di zucchero
Se ho >6 persone: 50 grammi di zucchero.

Questo esempio è *INCOMPLETO MA CONSISTENTE*: infatti, non si sa quanto zucchero usare se ci sono 5 persone.

Se ho >4 persone: 10 grammi di zucchero
Se ho >6 persone: 50 grammi di zucchero

Questo esempio è *COMPLETO MA INCONSISTENTE*: per 7 persone non so quale regola seguire.

Finitezza: produce i risultati in un numero finito di passi. Questa proprieta' puo' essere definito tramite un criterio di arresto o altro, mescolare finche' non e' fuida….. La divisione di un numero (numero delle cifre, la precisione). Logicamente l'algoritmo deve fornire i risultati in un *NUMERO FINITO DI PASSI.*

Molto spesso le operazioni da fare possono essere anche di tipo ciclico, cioe' possono essere ripercorse piu' volte; in questo caso si parla di *LOOP* o *ITERAZIONE* ma l'importante e' che comunque il numero di operazioni sia finito. Cosi' un loop o un'iterazione infinita inserita in un programma porta all'esecuzione delle stesse istruzioni in modo indefinito. In questi casi un computer "si perde" nelle stesse istruzioni nel tentativo di giungere alla fine. Devo quindi stabilire dei *CRITERI DI ARRESTO* per sapere quando ho ottenuto il risultato finale. In caso contrario il calcolatore continuerebbe all'infinito.
Un controllo di processo di un reattore nucleare non deve mai finire pertanto non deve avere questa proprieta'.

Liveness, Vivezza: capacita' di mantenersi in esecuzione

Efficienza: produrre i risultati nel modo piu' veloce possibile ma comunque corretto.
Una proprietà fondamentale è appunto l'efficienza: infatti l'algoritmo deve saper produrre i risultati in *MANIERA CORRETTA* più velocemente possibile.
Per esempio, l'algoritmo riguardante il funzionamento e la gestione di una comune macchinetta per il caffe' non puo' essere strutturato in modo che l'utente debba attendere piu' di tre minuti per l'erogazione del caffe' altrimenti l'utente pensa che la macchinetta sia guasta e magari la prende a calci con ovvie conseguenze.

Decidibile: non esiste un punto in cui la strada/scelta da prendere/da fare e' data/prodotto dalla casualità: in base alle informazioni iniziali deve essere chiaro che cosa deve essere eseguito.

Se c=3 allora fai A o B (e' indecidibile)

Esempi di algoritmi indicibili possono essere:

"ricercare il percorso che collega due città italiane"

Se non sono definiti *CRITERI DI DECISIONE* non si sa come comportarsi se ci trova di fronte ad un bivio. L'unica possibilità è *PROCEDERE IN MODO CASUALE*: solo in questo modo si può avere in uscita una *SOLUZIONE UNICA*. La decidibilità dipende sia dai dati iniziali che dall'algoritmo.

Per esempio se c'è la possibilità di cucinare A o B, il calcolatore non è in grado di prendere una decisione senza procedere per via casuale. Nell'esempio della ricetta: Mettere miele o zucchero.

Deterministico: il risultati prodotti dall'algoritmo sono ripetibili e determinati in modo univoco in modo che essi possano essere valutati da chi ha sviluppato l'algoritmo come giusti o sbagliati, accettabili o meno. Quando un algoritmo non e' decidibile può produrre risultati non deterministici. Una torta bruciata e' un esempio di un algoritmo che ha portato ad un risultato errato.

Risolubilita' al calcolatore devono essere posti problemi risolubili, perché in tal caso ci sarà sempre un algoritmo per risolvere.

Esempio: “Trovare 4 numeri interi per cui valga $x^n + y^n = z^n$, n>z “ . Si può vedere che questa relazione non ha soluzioni per n>2 , e quindi per questi valori l’algoritmo non è più risolubile !

Polimorfico: l’algoritmo puo’ lavorare su dati diversi come se questi fossero uguali, ossia esso specifica come una stessa funzione debba comportarsi su oggetti diversi con proprieta’ specifiche. Esempi di algoritmi polimorfici possono essere: ”Disegnare le figure geometriche; Impastare diversi tipi di pasta ma con lo stesso procedimento; Fare il pieno alla macchina di benzina o gasolio”.

In questo modo aumenta la possibilita’ di riutilizzare lo stesso algoritmo se deve essere applicato a contesti leggermente diversi da quello originale.

Definitezza, NON AMBIGUITA' , i passi che descrivono l'algoritmo devono essere non ambigui e non ambiguita’ significa appunto che il significato delle istruzioni dell’algoritmo deve essere sempre lo stesso qualunque sia la persona che lo sta eseguendo.
Percio’ tutte le possibili condizioni che possono verificarsi devono essere previste e per ciascuna di esse devono essere specificate le opportune azioni.

ESEGUIBILITA' deve essere possibile eseguire i passi in modo automatico. Molto importante visto che informatica significa informazione automatica.
L'eseguibilita' può essere automatica o manuale. Manuale se con l'aiuto dell'uomo, automatica se completamente automatizzabile ed eseguibile da una macchina.

1.3 Variabili e Costanti

Tutti i dati che si trovano all'interno di un algoritmo/programma possono essere suddivisi in costanti e variabili. Tutti i numeri, cifre, caratteri il cui valore rimane immutato durante lo svolgimento di un algoritmo, anche dopo che sono state trovate le soluzioni, prendono il nome di *COSTANTI* proprio perché il loro valore rimane invariato indipendentemente dall'algoritmo.

Al contrario sono delle *VARIABILI* tutti quei caratteri che assumono valori diversi in base ai dati iniziali inseriti dall'utente. In questo modo è possibile sfruttare l'algoritmo con dati iniziali diversi.
In parole semplici una variabile e’ un elemento paragonabile a un contenitore dentro il quale si puo’ inserire di tutto durante l’esecuzione dell’algoritmo.

Se torniamo all’esempio del calcolo delle radici di un polinomio di 2^0 grado, è possibile dire che nell’espressione:

$$\Delta = b^2 - 4*a*c$$

- 4 e 2 sono *COSTANTI*, mentre
- b, a, c, Δ, x1, x2, sono *VARIABILI , in questo caso sono numeri reali* (e di conseguenza anche $x_{1,2}$)

Quando e' stato formalizzato l'algoritmo per il calcolo delle radici l'espressione trovata per le radici le definisce in modo esatto cioè con un numero infinito di cifre dopo la virgola, cioè di decimali. In effetti il calcolatore non e' in grado di lavorare con un numero infinito di cifre (l'algoritmo sarebbe non finito) oppure lasciare indicata la divisione.

L'elaboratore e' uno strumento digitale e pertanto lavora con numero finito di cifre decimali sia nel caso di numeri interi che reali.

Il problema della soluzione dell'equazione di secondo grado dovrebbe essere riformulato per esempio come:
"trovare le radici del polinomio di secondo grado con una precisione maggiore di 1/10000"

1.3.1 Tipi di variabili e di costanti

Tutte le variabili sono caratterizzate da un nome, l'identificatore,e dal valore che possono assumere attraverso "istruzioni" dette di assegnamento che modificano il contenuto della cella.Si possono avere vari tipi di variabili:

- **BOOLEANO**:i dati di tipo booleano assumono i valori "vero" o "falso" e possono essere memorizzati internamente con un singolo bit al quale si puo' assegnare il valore "0" o "1".
- **INTERO(**integer):i dati di tipo intero possono assumere qualunque valore intero,positivo o negativo:su di essi sono definiti gli operatori aritmetici.

Per quanto riguarda le costanti,ossia i valori fissi esplicitamente scritti nei programmi,esse si dividono in:
COSTANTI

- **NUMERICHE** la cui forma e' quella classica dei numeri.
- **DATI TESTUALI** il cui valore viene racchiuso tra apici.Spesso e' possibili anche scrivere "stringhe" di caratteri,cioe' sequenze di codici ASCII (argomento che verra' esaminato piu' tardi nel capitolo successivo).

1.4 L'Elaboratore Elettronico Digitale

Fino a questo momento si e' accennato all'elaboratore senza però definire cos'è. L'elaboratore è quella "macchina" in grado di eseguire gli algoritmi forniti dall'utente sulla base di dati diversi ma in modo automatico. Una volta che fornito un algoritmo dall'esterno, in molti casi l'elaboratore è in grado di lavorarci sopra molto più rapidamente rispetto all'uomo. Sull'elaboratore l'utente può eseguire programmi diversi tra loro per tipologia e per formulazione, ma è necessario che venga usato un *LINGUAGGIO DI PROGRAMMAZIONE* compatibile con l'elaboratore stesso. Infatti, l'elaboratore è in grado di operare solo in base ad istruzioni redatte in un formato per lui eseguibile. Un algoritmo deve percio' essere descritto attraverso un linguaggio generalizzato costituito da strutture linguistiche definite. Solitamente le proposizioni che vengono utilizzate per gli algoritmi contengono sia una descrizione delle operazioni che devono essere eseguite, sia una descrizione degli oggetti sui quali si devono effettuare le operazioni per ottenere i risultati voluti. L'elaboratore e' flessibile perché permette di eseguire algoritmi diversi semplicemente cambiando il programma.
Pertanto con programmi diversi verranno utilizzati dati diversi o uguali e producono risultati diversi e/o uguali ma sempre coerenti con il programma.

I programmi possono essere scritti in *LINGUAGGIO MACCHINA o in linguaggi di alto livello, come vedremo.*

In parole semplici l'elaboratore e' una apparecchiatura AUTOMATICA, DIGITALE o ELETTRONICA capace di effettuare trasformazioni su dei dati in ingresso. Si dice elettronico quando l'ELABORATORE e' costruito con tecnologia elettronica. Mentre e' digitale quando lavora utilzzando una logica discreta caratterizzata da un precisione finita e da un numero finito di rappresentazioni.

E' necessario specificare che un elaboratore digitale, oltre a richiedere un dominio sui dati di ingresso e sull'applicazione, lavora sempre con numeri a *CIFRE FINITE*. Questo porterà in seguito ad approssimazioni nella rappresentazione di numeri binari in virgola mobile.
Questa "limitazione" è data dal fatto che i calcolatori sono *SISTEMI DIGITALI*, non analogici, e quindi forniscono sempre una risoluzione finita (0/1). Tuttavia, anche se c'è una perdita in termini di precisione, con i sistemi digitali è possibile lavorare su numeri ben definiti.

Per essere eseguibile un algoritmo deve essere codificato in un linguaggio comprensibile per chi lo esegue.
Gli elaboratori sono in grado di eseguire istruzioni se queste sono opportunamente codificate.
Un insieme di istruzioni ordinate secondo un certo schema che sono l'implementazione di un algorithmo puo' essere chiamato programma.

1.5 L'Informatica

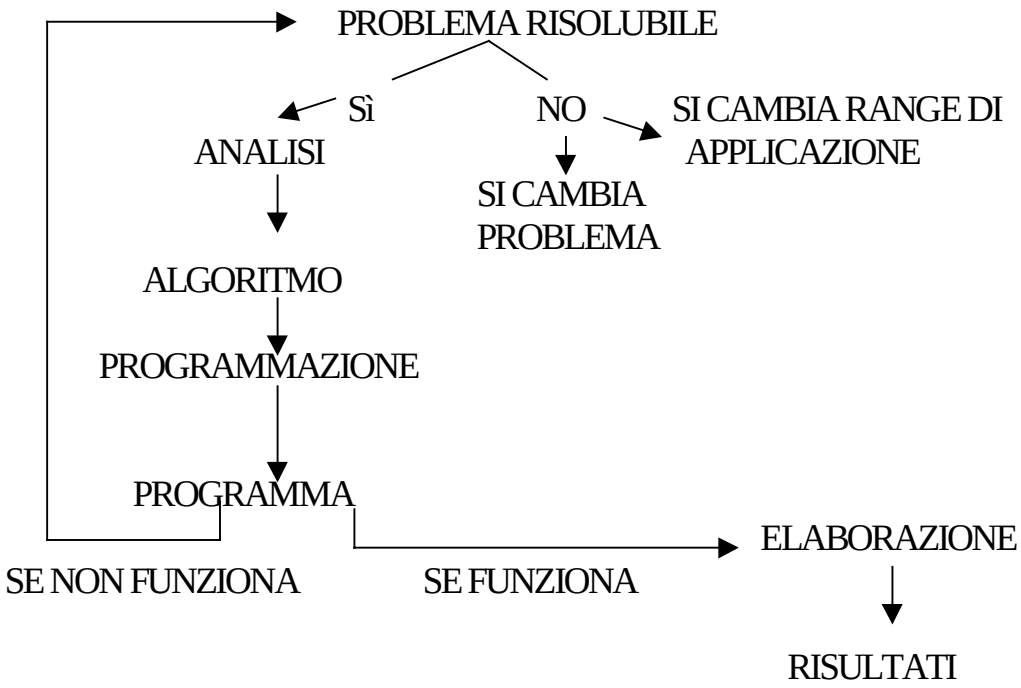
Puo' essere vista come la forma contratta di *INFORMAZIONE AUTOMATICA*. In generale è quella scienza che studia tutti i modi per poter manipolare l'informazione e codificare gli algoritmi secondo precisi automatismi. Quindi si può dire che il problema principale è quello di trovare una rappresentazione che conduca nella maniera più lineare e precisa ad una produzione informazione automatica di certi risutlati. In questo caso l'elaboratore deve eseguire una determinata successione di operazioni senza interventi esterni, una volta predisposto e avviato ad eseguirla.

-Non e' la scienza e la tecnica dei microprocessori dei calcolatori
-Non e' la scienza dell'informazione rivolta solamente ad un gruppo di tecnici altamente specializzati
-Non e' la teoria dell'informazione

Problema ⟶ Algoritmo ⟶ Programma

L'informatica e' lo studio sistematico degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, l'analisi, il progetto, le proprieta', la realizzazione e la loro applicazione a casi reali.

Per passare dal problema al programma si deve applicare una serie di procedimenti suddivisi in "analisi" e "programmazione": l'insieme di queste attività serve per risolvere il problema tramite l'elaboratore.



Lo scopo dell'analisi è definire un algoritmo, cioè un elenco finito di istruzioni che determinano una serie di operazioni indispensabili per ottenere una soluzione al problema. Prima ancora di formulare l'algoritmo si deve analizzare il problema e definire se è risolvibile o meno, e se i dati rientrano nel dominio stabilito. Il tutto in base ai presupposti ed ai dati iniziali.

Se il problema è risolvibile si passa alle operazioni di analisi e posso definire l'algoritmo. Se non fosse risolvibile ci sono due alternative: si può decidere di cambiare completamente problema, oppure si può ridurre il range di applicazione e controllare nuovamente la risolubilità del problema.

Per esempio, nella relazione

$$x^n + y^n = z^n \qquad \text{con } n>z$$

non si potevano avere soluzioni per n>2: in questo caso l'unica possibilità era di limitare il range di applicazione imponendo n =<2 .

Una volta definito l'algoritmo si passa alla parte della "programmazione", che ha per scopo la definizione di un programma ottimale. Non è detto però che il programma funzioni, ovvero che l'algoritmo trovato produca qualche soluzione: in questo caso bisogna ritornare al problema iniziale, modificarlo se necessario, e ripetere tutte le operazioni dall'inizio.

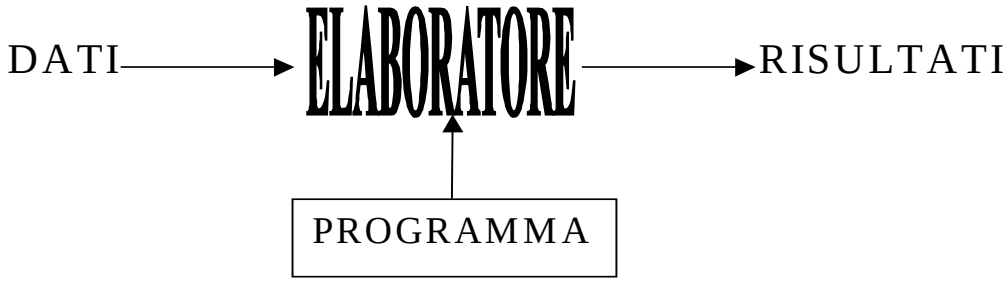
1) Analisi del problema
2) ipotizzo una soluzione (algoritmo)
3a) verifiche di consistenza e di completezza
3b) provare la soluzione rispetto a vari casi tipici, oppure test esaustivo
4) se non esiste una soluzione allora torna a (1)

5) prova a dimostrare che non esiste soluzione se vero smetto e spendo meglio il mio tempo
se non vero riprovo da (1)

1.6 Programmi e Linguaggi

E' l'insieme di algoritmi necessari per arrivare alla soluzione del problema. Inoltre il programma è quello strumento che permette di controllare il regolare procedimento delle singole istruzioni dell'algoritmo (se c'è un solo algoritmo) oppure decide, in base a criteri logici e matematici, in quale sequenza cronologica applicare i diversi algoritmi.

-*IL PROGRAMMA* è una traduzione dell'algoritmo in un linguaggio comprensibile dal calcolatore ed deve essere in grado di produrre il risultato o di dichiarare che la soluzione non e' stata trovata. Cambiando il programma registrato nella memoria dell'elaboratore si puo' mettere l'elaboratore in grado di risolvere problemi di natura diversa;l'elaboratore diventa quindi una apparecchiatura universale dal momento che puo' essere usata per risolvere tutti quei problemi la cui soluzione puo' essere descritta sotto forma di programma.E' pero' assolutamente necessario che la sintassi delle singole righe del programma sia assolutamente corretta,poiche' anche la mancanza di un punto e virgola blocca l'esecuzione visualizzando un messaggio di errore.Tale rigorosita'deriva dal fatto che le istruzioni devono essere univoche e formalmente corrette:da qui deriva anche il termine di linguaggi formali con i quali si identificano spesso i linguaggi di programmazione.



Prima di cominciare la stesura dell'algoritmo deve essere ben chiaro il *DOMINIO DEI DATI DI INGRESSO* . Se i dati vengono presi al di fuori del dominio non hanno più un senso e una validità. Quando inserisco dei dati devo definire un *RANGE*, un campo ristretto al di fuori del quale ogni dato perde di significato. Le informazioni disponibili devono essere limitate.
DINAMICA (RANGE): dice qual è il numero più grande rappresentabile (Es. $\pm 10^{-6}$)
PRECISIONE: è il numero più piccolo rappresentabile (Es. $\pm 10^{-12}$), ovvero l'attendibilità con cui un risultato può avvvicinarsi a quel numero (rifacendosi all'analisi matematica si può parlare di "intorno").

DOMINIO DELL'APPLICAZIONE: quando viene posto un problema devono essere definiti i termini dei dati dell'algoritmo e dei suoi risultati, il dominio dell'applicazione e' l'insieme del dominio dei dati più la descrizione formale della classe dei problemi che l'algoritmo e' in grado di risolvere.

Es: tutte le equazioni di secondo grado
Es: le equazioni di secondo grado che hanno delta>0

DOMINIO DELL'APPLICABILITA': è il range che viene imposto sui dati dell'algoritmo (Es. nel calcolo delle radici di un polinomio di secondo grado viene imposto il $\Delta >= 0$).

2 Algebra di Boole

I fondamenti dell'algebra Booleana sono stati delineati dal matematico inglese George BOOLE, nato a Lincoln nel 1815 e morto nel 1864, in un lavoro pubblicato nel 1847 riguardante l'analisi della logica e in particolare l'algebra della logica. Questa algebra include una serie di operazioni che si effettuano su delle variabili logiche, dette appunto variabili Booleane: quantita' che permettono di codificare le informazioni su due soli livelli. Nell' algebra di Boole essendo, a differenza di quella tradizionale, un' algebra binaria, i numeri possono assumere soltanto due *stati* :

- 0/1;
- v/f (vero, falso);
- l/h (low, high);
- t/f (true, false);
- on/off; (acceso/spento)
- T,⊥ (true/false, vero/falso)

Usualmente si usa la notazione 0/1.

Su questi simboli si devono anche definire degli operatori e delle regole, che governano le operazioni, che ci permettono di utilizzarli:

- operatori prodotto logico and (∧ , ∘, &&, ∗) ,
- somma logica or (∨,+, ||),
- negazione o complementazione not (### , -, ~, sovrascritto, !),

Una proposizione, un costrutto linguistico, del quale si puo' dire se e' vero o falso, diventa un predicato quando in essa compare una variabile e il valore delle variabili determina il valore di verita' della proposizione stessa. Valutare il predicato significa quindi eseguire quell'operazione che ci consente di determinare la verita' o la falsita' del predicato stesso: i due valori "vero" o "falso" sono detti valori logici. I predicati che contengono un solo operatore sono detti predicati semplici mentre quelli composti sono caratterizzati dai simboli NOT, AND, OR.

Esempio: A>3 and B or D

2.1 Gli operatori di confronto

Gli operatori di confronto vengono usati per mettere a confronto due valori. Essi sono indicati nel modo seguente:

== uguale a **< > diverso da**
> maggiore di **>= maggiore o uguale a**
< minore di **<= minore o uguale a**

Il risultato del confronto e' indicato da 1 se e' vero, da 0 se e' falso.

2.2 Gli operatori Booleani: AND, OR, NOT

2.2.1 Operatore AND

L'operatore **and**, detto anche congiunzione logica, è definito come l'operatore che definisce A $\wedge$ B vero quando sia A che B sono veri.

es: <L'amplificatore funziona se è alimentato e ben costruito>

E' chiaro che debbono manifestarsi contemporaneamente due condizioni per il funzionamento dell'amplificatore:

1. deve essere alimentato
2. deve essere ben costruito

Assegnando opportunamente alle variabili binarie i valori <0> e <1> si può scrivere:

Y= (<1> l'amplificatore funziona <0> l'amplificatore non funziona)
A= (<1> è alimentato <0> non è alimentato)
B= <1> è ben costruito <0> non è ben costruito)

E' possibile ora compilare una tabella, detta ***tabella della verità*** ,che descrive il valore del risultato dell'operazione in funzione del valore degli operandi in cui compaiono tutte le 2^n possibili configurazioni delle variabili indipendenti, con *n* numero delle variabili.

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

La tabella è in pieno accordo con l'equazione algebrica Y=A$*$B in quanto Y=1 (l'amplificatore funziona) se e soltanto se A e B valgono 1 (c'è alimentazione e l'amplificatore è ben costruito).

Es: 101101 and 010110 Le operazioni vanno effettuate considerando i bit uno per volta

```
101101 and
010110=
----------
000100
```

2.2.2 Operatore OR

L'operatore **or** ,detto anche disgiunzione logica, definisce A v B vero quando A o B sono veri; la o ha carattere sia esclusivo (o questo o quello) sia carattere inclusivo (o questo o quello o entrambi).

A	B	A v B
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

Es:101110 or 010010=

101110 or
010010=

111110

2.2.3 Operatore NOT

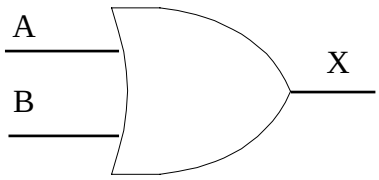
A differenza degli altri operatori, l'operatore not é unario in quanto la sua operazione riguarda un singolo elemento .
La sua funzione é di far cambiare lo stato di un elemento, commutando l'elemento 1 in 0 e viceversa.

A	-A
0	**1**
1	**0**

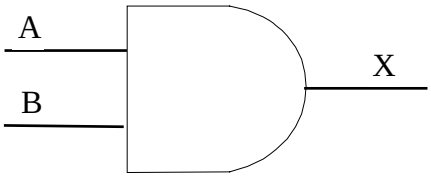
Es: -(101110)=010001

2.2.4 And, Or, e NOT e i Circuiti

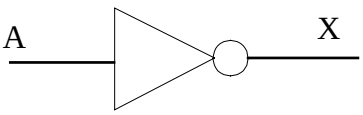
Da un punto di vista fisico le operazioni logiche dell'algebra booleana possono essere realizzate con dedei dispositivi elettronici noti come **porte logiche o circuiti elementari.**



X=A or B

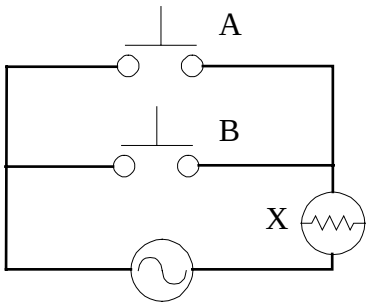


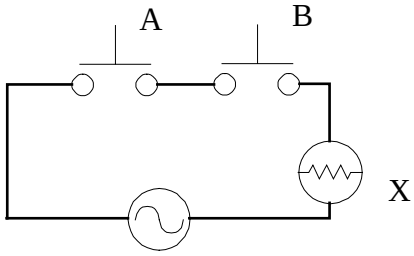
X=A and B



X = not A

Esempi di circuiti con le lampadine relativi alle operazioni AND e OR





2.2.5 Implica e Coimplica

A -> B A IMPLICA B : il termine implica non significa che se A
e' vero (o falso) allora necessariamente B e' vero (o falso).Come e' possibile vedere dalla seguente tabella della verita' A -> B e' equivalente a -A or B.

A	B	-A	-A or B
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

A <-> B A COIMPLICA B e' equivalente a A -> B and B->A

A	B	-A	-B	-A or B	-B or A	-AorB and –B
0	0	1	1	1	1	1
0	1	1	0	1	0	0
1	0	0	1	0	1	0
1	1	0	0	1	1	1

2.2.6 Proprietà degli operatori AND e OR

or	and
xv1=1	**$x_{\wedge}$1=x**
xv0=x	**$x_{\wedge}$0=0**
xvx=x	**$x_{\wedge}$x=x**
xv-x=1	$x_{\wedge}$-x=0

X significa don't care cioe' non ha importanza.

2.3 Teoremi dell'Algebra di Boole

L'and e l'or godono delle proprieta' commutativa, distributiva ed associativa.

1) Per ogni X,Y XvY= YvX
X^Y = Y^X

2)Per ogni X,Y,Z X + (YZ) = (X +Y) (X +Z)
X (Y+Z) = (XY) + (XZ)

3)Per ogni X,Y,Z X + (Y + Z) = (X + Y) + Z
X (YZ) = (XY) Z

Inoltre vi sono le seguenti proprietà:

1) X+XY=X (assorbimento 1)
2) X+(-X)Y=X+Y (assorbimento 2)
3) XY+YZ+(-X)Z=XY+(-X)Z (assorbimento 3)
4) -(X+Y)=(-X)(-Y) **DE MORGAN**

Sono valide anche le duali

1) X(X+Y)=X
X+XY=X
-(X+XY)=(-X)
(-X)(-(XY))=(-X)
(-X)((-X)+(-Y))=(-X)
X(X+Y)=X

2) X((-X)+Y)=XY
X+(-X)Y=X+Y
(-X)+X(-Y)=-(X+Y)
(-X)(X+(-Y))=(-X)(-Y)
X((-X)+Y)=XY

3) (X+Y)(Y+Z)((-X)+Z)=(X+Y)((-X)+Z)

4) -(XY)=(-X)+(-Y)

Assorbimento 1) X+XY=X
L'assorbimento 1) si può dimostrare per esempio per mezzo della tabella della verita':

X+XY = X
Si puo' riscrivere come X(1+Y) = X ma 1+Y=1 che implica X=X

X	Y	XY	X+XY
0	0	0	0
1	0	0	1
0	1	0	0
1	1	1	1

Assorbimento 2)X+(-X)Y=X+Y

X(X+(-X)Y) = XX+XY
XX+X(-X)Y = XX + XY
X = X+XY

come prima

X	**Y**	**(-X)**	**(-X)Y**	**X+Y**	**X+(-X)Y**
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

Questo assorbimento si puo' dimostrare anche da un punto di vista matematico; pertanto :
X+(-X)=X+Y
X(X+(-X)Y)=XX+XY

Per la proprieta' distributiva si ha:
XX+X(-X)=XX+XY;

inoltre ricordando che XX=X e X(-X)=0 si ottiene X=X+XY (ASSORBIMENTO 1)

Assorbimento 3)XY+YZ+(-X)Z=XY+(-X)Z

Z	**X**	**Y**	**(-X)**	**XY**	**YZ**	**(-X)Z**	**XY+YZ+(-X)Z**	**XY+(-X)Z**
0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	1	1
1	0	0	1	0	0	1	1	1
1	0	1	1	0	1	1	1	1
1	1	0	0	0	0	0	0	0
1	1	1	0	1	1	1	1	1

Teorema di De Morgan 4)-(X+Y)=(-X)(-Y)

X	Y	(-X)	(-Y)	X+Y	-(X+Y)	(-X)(-Y)
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Duale del Teorema di De Morgan -(XY)=(-X)+(-Y)

X	Y	-X	-Y	XY	-(XY)	(-X)+(-Y)
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

Come appare dal teorema di De Morgan e' superflua la scelta delle funzioni OR,AND,NOT come funzioni primitive :infatti l'operatore AND puo' essere espresso in funzione delle operazioni OR e NOT; analogamente l'operazione OR puo' essere espressa tramite AND e NOT. Pertanto e' facile verificare come l'espressione A>= OR B<0 equivalga a NOT(A<0 AND B>=0).

Infatti basta ricordare il teorema nella forma –(X and Y)= (-X) or (-Y).

A >= or B<0 Equivale a Not (A<0 and B>=0)

3 Sistemi di Numerazione

3.1 Sistemi di Numerazione Posizionali

Per definire un sistema di numerazione si ha bisogno:

1. di una base, b
2. di un insieme ordinato di cifre, I{}, distinte l'una dall'altra con dimensione pari a quella della base: dim(I) = b, L'insieme contiene simboli che hanno una certa posizione nella base ordinata.
3. di un codice di interpretazione cioè di un insieme di regole che permettono di determinare quale sia il numero rappresentato da un gruppo di cifre,
4. di un insieme di regole e di algoritmi per definire le operazioni fondamentali.

es: $5b^3+3b^2+2b+4$ per esempio dove b=10 base decimale in cui l'insieme delle cifre e': 0,1,2,3,4,5,6,7,8,9.
Si hanno unita', decine, centinai, etc.

Si possono usare anche altre basi

Booleana b=2 con [0,1], [V,F], [On, Off], etc.

Oppure base 5: b=5 con [0,1,2,3,4] [x,y,z,t,q]

Usando per esempio y,z q, l'espressione seguente yb^2+zb+q con b=5 diventa :

y*25+z*5+q = 25+10+4=39

Ovviamente basi più piccole hanno potenza espressiva minore.

3.1.1 Base generica b, numerazione posizionale

Un sistema di numerazione in base b è un sistema di numerazione posizionale la cui base è b. Un numero in base b è un numero rappresentato utilizzando la numerazione in base b ed è una sequenza ordinata di cifre cosi' rappresentata:

$$(C_{n-1}C_{n-2}.....C_1C_0 \, . \, C_{-1}C_{-2}.....C_{-m})_b$$

dove il simbolo "." è detto **punto di separazione o punto radice** e separa tra di loro le parti intera e frazionaria rappresentate la prima dalle *n* cifre $C_{n-1}.....C_0$ e la seconda dalle *m* cifre $C_{-1}.....C_{-m}$.
In effetti i coefficienti C rappresentano solo la posizione del simbolo nella base ordinata e non il valore della cifra. Questo risulta chiaro quando si realizzano basi con simboli e non con numeri ordinati.

Per esempio se si prende (tyq) in base 5 i C sono rispettivamente 3, 1, 4.

Nella parte intera il peso associato alla cifra C_0,cioe' quella piu' a destra, è $b^0=1$, il peso associato alla cifra C_1 è $b^1=b$ e cosi via fino ad arrivare alla cifra C_{n-1} alla quale è associato il peso b^{n-1}.

Per quanto riguarda la parte frazionaria, si parte dal peso b^{-1} , che è quello associato alla prima cifra a sinistra C_{-1}, perche' compare subito dopo il punto di separazione ,fino ad arrivare al peso b^{-m} associato all'ultima cifra a destra C_{-m}.
Qualunque sia il numero rappresentato, la cifra più a sinistra è la cifra più significativa del numero , la cifra più a destra è quella meno significativa. Una cifra e' quindi piu' significativa di un'altra se la sua posizione nel numero e' piu' a sinistra della posizione dell'altra cifra.

Studieremo esclusivamente i sistemi di numerazione posizionali (per ogni posizione un certo peso).
Tutti i sistemi di numerazione che si basano su un codice posizionale, possono essere scritti in forma polinomia:

$C_n\, b^n + C_{n-1} b^{n-1} + ... + C_0 \,.\, C_{-1} b^{-1} + ... + C_{-m}\, b^{-m}$

Per esempio nel caso di numeri decimali si ha:

1246, che significa che n=3, C3=1, C2=2, C1=4, C0=6

$1\ 10^3 + 2\ 10^2 + 4\ 10 + 6$ in cui ad ogni posizione nella sequenza e' associato un certo peso.

Nel caso invece di numeri in base 5 con, ad esempio, il seguente sistema ordinato di cifre :[A,B,C,D,E], l'insieme dei seguenti simboli E D D A corrispondono al numero decimale 590.

Si ha infatti: E x 125 + D x 25 + D x 5 + A x 1 =590

3.2 Rappresentazione dei Numeri Binari in Virgola Fissa

Per quanto riguarda la rappresentazione dei numeri, possono essere usati più codici binari a seconda del tipo di informazione numerica considerata (ad esempio, numeri frazionari o numeri interi).
La rappresentazione binaria in virgola fissa (o rappresentazione fixed point) consiste nel rappresentare ciascun numero assumendo una posizione prefissata del punto di separazione.
In particolare essa é usata per rappresentare i numeri interi con la convenzione di porre il punto di separazione alla destra del bit meno significativo.
La rappresentazione in virgola fissa utilizza una quantità prefissata di memoria ; ad esempio , nei processori intel 80386 e 80387 possono essere usate 1,2 oppure 4 voci di 16 bit.

Il sistema di numerazione binario si avvale di due simboli 0 e 1; ogni singola cifra di un numero binario prende il nome di bit (binary digit):

$$a_{n-1}\, 2^{n-1} + a_{n-2}\, 2^{n-2} + ... + a_1\, 2 + a_0 + a_{-1}\, 2^{-1} + ... + a_{-m}\, 2^{-m}$$

dalla forma polinomia si vede che i pesi associati alle cifre sono potenze del 2.

Un numero binario si legge un bit alla volta:
es.
$(10110)_2$ si legge uno zero uno uno zero

Dato un numero decimale intero e' inoltre sempre possibile calcolare il numero di bit necessari nella rappresentazione del corrispondente numero binario. Il massimo numero rappresentabile con "n" bit e' quel numero con "n" bit tutti uguali ad 1. Rifacendosi alla forma polinomia, ogni numero binario di tale forma si puo' scrivere nella seguente quantita' decimale:

$$\sum_{h=0}^{n-1} 2^k = 2^n - 1$$

Se ho ad esempio a disposizione 3 bit il piu' grande numero rappresentabile e' 7 perche' $2 - 1 = 2 - 1 = 7$. Si e' cosi' introdotto il concetto di dinamica di rappresentazione che verra' ripreso piu' accuratamente alla fine del paragrafo.

Per la conversione da binario a decimale si utilizza la forma polinomia:

es. $(101011.011)_2$ si può scrivere come

$$1\,2^5+0\,2^4+1\,2^3+0\,2^2+1\,2^1+1\,2^0+0\,2^{-1}+1\,2^{-2}+1\,2^{-3}=$$
$$32 + 8 + 2 + 1 + 0.25 + 0.125 =$$
$$(43.375)_{10}$$

Per la conversione da decimale a binario si devono trattare separatamente la parte intera e la parte frazionaria di un numero:

1) per la parte intera si utilizza il metodo delle divisioni successive; si divide il numero per 2 e il resto rappresenta la cifra meno significativa in binario; si divide poi per due il quoziente ottenuto ed il resto rappresenta la seconda cifra dal punto, radice e così si procede fino ad avere il quoziente zero

Dimostrazione: si vuole convertire il numero intero N in base 2, trovando quindi una forma polinomia tale che:

$$N= a_{n-1}\,2^{n-1}+a_{n-2}\,2^{n-2}+.....+a_1\,2+a_0$$

In effetti posso scrivere il numero in questo modo:

$N = Q\,b + R$

dividendo entrambi i termini per la base 2 una prima volta si ottiene

$$N/b = Q + R/b \Rightarrow \qquad N/2 = Q + R/2 = a_{n-1}\,2^{n-2}+a_{n-2}\,2^{n-3}+.....+a_1+(a_0 / 2)$$

e quindi il quoziente $Q = a_{n-1}\,2^{n-2}+a_{n-2}\,2^{n-3}+.....+a1$
mentre a0 e' il resto R. della prima divisione (e la cifra meno significativa della forma polinomia)
Dividendo ora Q nuovamente per 2 si ottiene un nuovo quoziente Q1 e un nuovo resto R1:
$Q/2=a_{n-1}2^{n-3}+a_{n-2}2^{n-4}+\ldots\ldots+a_1 2^{-1}$
Continuando a dividere per 2 i quozienti via via ottenuti , si arriva a determinare tutte le cifre binarie.

es. Si vuole convertire in binario il nu 43,

		resto	
43 / 2 =	21	1	$=a_0$
21 / 2 =	10	1	$=a_1$
10 / 2 =	5	0	$=a_2$

$5 / 2 = 2 \quad 1 \quad = a_3$
$2 / 2 = 1 \quad 0 \quad = a_4$
$1 / 2 = 0 \quad 1 \quad = a_5$

segue che $(43)_{10} = (101011)_2$ (N.B. Per la determinazione del numero binario i resti delle varie divisioni vanno scritti nell'ordine inverso rispetto a come sono stati calcolati).

2) per la parte frazionaria si utilizza il metodo dei prodotti successivi ;nella prima operazione si moltiplica per 2 il numero da convertire, nelle successive si moltiplica per 2 la parte frazionaria del risultato della moltiplicazione precedente. Le cifre del numero binario sono costituite dalla parte intera del prodotto della moltiplicazione precedente(la prima cifra che si ottiene è la più significativa).

Dimostrazione: si vuole convertire il numero frazionario F in base 2, trovando quindi una forma polinomia tale che:

$$F = a_{-1} 2^{-1} + a_{-2} 2^{-2} + \ldots\ldots\ldots + a_{-m} 2^{-m}$$

moltiplicando entrambi i termini per la base 2 una prima volta si ottiene

$$F\,b = R + F' \quad \Rightarrow \quad F*2 = R + F' = a_{-1} + a_{-2} 2^{-1} + \ldots\ldots\ldots + a_{-m} 2^{-m+1}$$

In cui a_{-1} rappresenta il valore della parte intera della moltiplicazione.
Moltiplicando adesso la parte frazionaria F1 si ottiene :
$F1x2 = a_{-2} + a_{-3} 2^{-1} + \ldots\ldots + a_{-m} 2^{-m+2}$

in cui a_{-2} e' la seconda cifra binaria; continuando su questa strada si determinano tutte le altre cifre binarie.

es. si vuole convertire 0.375 in binario

$0.375*2 = 0.750 \quad a_{-1} = 0$
$0.750*2 = 1.500 \quad a_{-2} = 1$
$0.500*2 = 1.000 \quad a_{-3} = 1$

segue che $(0.375)_{10} = (0.011)_2$

La maggior parte dei numeri frazionari non è rappresentabile in un numero finito di bit come nell'esempio precedente ,può conseguentemente risultare necessario fissare a priori il numero massimo di operazioni da eseguire

es. si vuole convertire 0.90 in binario si utilizzano un massimo di 6 bit

$0.90*2 = 1.80 \quad a_{-1} = 1$
$0.80*2 = 1.60 \quad a_{-2} = 1$
$0.60*2 = 1.20 \quad a_{-3} = 1$
$0.20*2 = 0.40 \quad a_{-4} = 0$
$0.40*2 = 0.80 \quad a_{-5} = 0$

ci si arresta dopo 5 passaggi , facendo quindi un'approssimazione

$(0.90)_{10}$ è circa $(0.11100)_2$

Il numero rappresentato risulta sicuramente minore di 0.90 dal momento che si eliminano addendi dalla forma polinomia

$$a_{-1}*2^{-1}+ a_{-2}*2^{-2}+ .....+a_{-5}*2^{-5}+ a_{-6}*2^{-6}+ a_{-7}*2^{-7}$$

Per riprova, facendo la conversione binario decimale si ottiene che

$(0.11100)_2=(0.890625)_{10}<(0.90)_{10}$

L'approssimazione per difetto del numero decimale dal quale siamo partiti, porta quindi a una condizione di errore valutabile come

Errore Assoluto E_A=Valore Vero V_V - Valore Rappresentato V_R
dall'esempio precedente si ottiene:

E_A= 0.90 (V_V)- 0.890625 (V_R)=0.009375

Errore Relativo $E_R=E_A/V_V$

dall'esempio : E_R=0.0104

Errore Percentuale $E_{\%}=E_R*100$

dall'esempio: $E_{\%}$=1.04%

l***'Errore Massimo*** dovuto all'approssimazione limitando il metodo dei prodotti successivi a k passaggi è:

$E_{max}=2^{-K}$ che risulta essere l'indice di **PRECISIONE** nella rappresentazione del valore in questione
.
Dal momento che trascuro tutti gli addendi della forma polinomia da $a_{-K}*2^{-K}$ in poi sicuramente

$$a_{-K-1}*2^{-K-1}+ a_{-K-2}*2^{-K-2}+ ..... <= a_{-K}*2^{-K}$$

cioè pari al peso dell'ultima cifra rappresentata.

Dall'esempio

$E_{max}= 2^{-6}=0.015625>0.009375(E_A)$

Non si commettono errori quando nel procedimento delle moltiplicazioni successive si ottiene come risultato un numero con la parte frazionaria nulla.

Ad esempio nella conversione del numero decimale 0.4375 in binario abbiamo:

0.4375	* 2 = 0.875	0
0.875	* 2 =1.75	1
0.75	* 2 = 1.5	1
0.5	* 2 =1.0	1

Quindi l'errore commesso e' 0.

Dato un numero finito N di bit con cui poter rappresentare in binario un numero intero è possibile determinare il numero più grande rappresentabile. Se ad esempio dispongo di 5 bit, il numero massimo rappresentabile è

$(11111)_2$ che in forma polinomia è $1*2^4+1*2^3+1*2^2+1*2^1+1*2^0$

Cioè si hanno 2^N combinazioni da 0 a 2^n -1. Il bit di peso massimo ha come peso 2^(n-1).

Il numero massimo rappresentabile è dato da

2^N-1 (***Dinamica di Rappresentazione***)

Viceversa dato un numero decimale D si può determinare il numero N di bit necessari per la rappresentazione del corrispondente numero binario; N è il più piccolo intero tale che

$2^N-1>D$

e quindi $N=\log_2(D+1)$, approssimando per eccesso all' intero successivo

es.

D=2073 $N=\log_2 2074=12$ $2^{12}=4096>2073$

11 bit non sarebbero stati sufficienti perché il numero massimo rappresentabile con questi è $2^{11}-1=2047<2073$ .Pertanto nella rappresentazione dei numeri in virgola fissa posso decidere quanti bits ho bisogno per la parte intera (che dinamica ho nella rappresentazione) e per la parte frazionaria (che precisione ho nella rappresentazione).

2 alla n	N	2 alla - n
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.015625
128	7	0.0078125
256	8	0.00390625
512	9	0.001953125
1024	10	0.0009765625
2048	11	0.00048828125
4096	12	0.000244140625
8192	13	0.0001220703125
16384	14	0.00006103515625
32768	15	0.000030517578125
65536	16	0.0000152587890625
131072	17	0.00000762939453125
262144	18	0.000003814697265625
524288	19	0.0000019073486328125
1048576	20	0.00000095367431640525

3.3 Operazioni fra numeri Binari

Ai numeri binari si possono applicare le 3 OPERAZIONI LOGICHE: AND,OR,NOT i cui risultati sono calcolati in base alle rispettive tavole della verita'.

Operatore **AND**

```
1 0 1 0 1 1 1 0
1 0 0 1 0 1 0 1
_______________
1 0 0 0 0 1 0 0
```

Questo particolare operatore puo' essere utilizzato per effettuare le cosidette "mascherature":il metodo consiste nel sommare ad numero qualsiasi, del quale voglio calcolare il valore di una sua certa cifra, un numero costituito da tutti 0, tranne che in corrispondenza di quella determinata cifra.
Ad esempio, se si desidera calcolare quanto vale la terza cifra di un numero qualsiasi(esempio 0110110), si potra' effettuare la seguente operazione:

```
0 1 1 0 X 1 1 0        AND
0 0 0 0 1 0 0 0
_______________
0 0 0 0 1 0 0 0
```

Se si ottiene un numero = a 0 allora la X =0, altrimenti se il numero e' diverso da 0 la X e' diversa da 0.

Operatore OR

```
1 0 1 0 1 1 0
1 0  1 0 1 0 0
_____________
1 0 1 0 1 1 0
```

Operatore NOT

```
1 0 1 1 0 1 0 1
_______________
0 1 0 0 1  0 1 0
```

3.2.1 Somma fra Binari

E' possibile definire l'operazione di somma tra due numeri binari A e B tramite la seguente tabella:

A	B	A+B	Riporto
0	0	0	/
0	1	1	/
1	0	1	/
1	1	0	1

Il **riporto** nasce dal fatto che 1 + 1 =2 ma nel sistema binario non esiste il simbolo 2 e quindi occorre riportare l'1 nella posizione seguente che ha valore 2. Percio'quando la somma é maggiore di 1, si deve effettuare il riporto di 1 sulla cifra immediatamente a sinistra. Es.

```
  1 1  1  1          Riporto        verifica decimale
   100110+                             38+
   011010=                             26=
 ________                            ____
  1000000                             64
```

3.2.2 Sottrazione fra Binari

Analogamente a quanto fatto per la somma, è possibile ricorrere a una tabella anche per la differenza.

Prestito	A	B	A-B
/	0	0	0
1	0	1	1
/	1	0	1
/	1	1	0

Se il minuendo è minore del sottraendo la sottrazione si effettua con il prestito di 1 dalla cifra immediatamente a sinistra (se anche questa è uguale a 0 si scorre a sinistra fino alla prima cifra uguale a 1): effettuando il prestito al posto dell'1 prestato rimane lo 0.

Esempi di differenze tra numeri binari:

```
          10  0  10 10                          Prestiti          0  1  1 10  1  1  1 10
1  0  1  1  0  1  1  0  1-                                        1  0  0  1  0  0  0  0-
1  0  0  0  1  0  1  1  0=                                              1  0  0  0  1=
-------------------------                                        ----------------
0  0  1  0  1  0  1  1  1                                           1  1  1  1  1  1  1
```

3.2.3 Moltiplicazione fra Binari

A	B	A*B
0	0	0
0	1	0
1	0	0
1	1	1

Tenuto conto che se il moltiplicatore è 1, il prodotto è uguale al moltiplicatore, e se il moltiplicatore è 0 il prodotto è 0, la moltiplicazione binaria si riduce, come quella decimale, ad una addizione di prodotti parziali.

Es.

```
     1001110 X 10101          Verifica decimale
     1001110                   78 X 21=1638
   1001110--
 1001110--
 ---------------
 11001100110          (11001100110)2 = (1638)10
```

3.2.4 Divisione fra Binari

La divisione binaria si svolge in modo analogo a quella decimale; procede quindi per sottrazioni tra parti del dividendo e divisore.

```
Es.   dividendo          divisore            verifica decimale
    11001100110 |  10101                 1638 : 21 = 78
                |__________
    10101          1001110               ( 1001110 )2 = ( 78 )10
    00100100
       10101
        11111
        10101
        010101
         10101
             0
```

Per la lezione:
Fare prima il prodotto di due numeri binari magari uno con virgola mobile

1101 x 11.01

il risultato potra' essere diviso per 1101 in modo perfetto, cioe' potra' essere preso come dividendo. Altrimenti il numero puo' risultare no divisibile e pertanto si puo' arrivare a un risultato che ha infinite cifre dopo la virgola.

Sui numeri binari esistono inoltre altre due operazioni : **SCORRIMENTO A DESTRA (SHR) e SCORRIMENTO A SINISTRA (SHL).**
Prendiamo ad esempio il numero binario 10101101 che corrisponde al numero decimale 173;operare uno scorrimento a sinistra significa trasferire tutti i bit del numero binario di un posto alla propria sinistra in modo che il bit meno significativo diventi = a 0. Otteniamo cosi' il numero 101011010 che corrisponde a $(346)_{10}$: come possiamo facilmente notare il numero e' stato semplicemente raddoppiato e percio' possiamo affermare che lo SHL corrisponde ad una moltiplicazione per 2.Eseguendo "n" scorrimenti a sinistra, eseguo "n" moltiplicazioni per 2 del numero in questione.
Al contrario lo SHR, che consiste nel far scorrere di un posto a destra ogni cifra del numero ,corrisponde ad una divisione per 2.

3.3 Numeri Ottali

Per convenzione i simboli utilizzati dal sistema ottale sono i seguenti:

0,1,2,3,4,5,6,7

nella forma polinomia il peso associato ad ogni cifra è una potenza dell'8; avremo quindi:

$$a_{n-1} *8^{n-1}+a_{n-2}*8^{n-2}+.........+a_1*8^1+a_0*8^0$$

La conversione da ottale a decimale si avvale della forma polinomia.
Ad esempio:

$$(1534)_8 = 1*8^3+5*8^2+3*8^1+4*8^0 = 512+320+24+4 = (860)_{10}$$

Per la conversione inversa si utilizza il metodo delle divisioni successive, dividendo per 8 se si tratta di un numero intero e moltiplicando per 8 (metodo delle moltiplicazioni successive) se si tratta di un numero frazionario.

		resto
5437 / 8 =	679	5 $=a_0$
679 / 8 =	84	7 $=a_1$
84 / 8 =	10	4 $=a_2$
10 / 8 =	1	2 $=a_3$
1 / 8 =	0	1 $=a_4$

segue che $(5437)_{10} =(12475)_8$

Per la conversione da numeri ottali a binari è necessario evidenziare le analogie esistenti tra forma polinomia ottale e forma polinomia binari.

La forma polinomia di un numero binario

$$a_{n-1}\,2^{n-1}+a_{n-2}\,2^{n-2}+.....+a_8\,2^8+a_7\,2^7+.....+a_2\,2^2+a_1\,2+a_0$$

può essere scritta raggruppando tre addendi alla volta

$$(a_8\,2^2+a_7\,2^1+a_6)\,2^6+(a_5\,2^2+a_4\,2^1+a_3)\,2^3+(a_2\,2^2+a_1\,2^1+a_0)\,2^0$$

si può osservare che:

- $2^0 = 8^0 = 1$
  $2^3 = 8^1 = 8$ (le potenze dell'8 sono coefficienti della forma polinomia ottale)
  $2^6 = 8^2 = 64$

- un coefficiente della forma polinomia ottale può essere espresso con 3 bit

.....+$(a_n\,2^2+a_{n-1}\,2^1+a_{n-2}\,2^0)\,2^{n-2}$+.....(forma polinomia binaria)

.....+(b_P) 8^p +.....(forma polinomia ottale)

per le conversioni tra numeri ottali e numeri binari si tenga presente la seguente tabella:

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

Es.

$(1\ 5\ 3\ 7)_8$
001 101 011 111

segue che $(1537)_8 = (001/101/011/111)_2$

Es. di conversione da binario a ottale in cui il numero viene suddiviso a gruppi di tre cifre,da destra verso sinistra,aggiungendo degli zeri nell'ultimo gruppo a destra se necessario e convertendo poi ciascun gruppo di tre cifre binarie nella sua corrispondente cifra ottale.

$(100/110/011/001)_2$
4 6 3 1

segue che $(100/110/011/001)_2 = (4631)_8$

Per la conversione della parte frazionaria, analogamente a quanto visto per la parte intera, si raggruppano tre addendi alla volta partendo dalla cifra più significativa e spostandosi verso destra.

Es.

$(0.101/110/010)_2 = (0.562)_8$

Come possiamo facilmente notare dagli esempi,la rappresentazione ottale e' piu' compatta dal momento che ad ogni cifra ottale corrispondono 3 bits:pertanto ogni numero ottale con "n" cifre rappresenta un numero binario di 3"n" cifre.

3.4 Numeri Esadecimali

I simboli esadecimali sono:

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

(i codici A, B, C, D, E, F, corrispondono in decimale rispettivamente ai numeri 10, 11, 12, 13, 14, 15).
La forma polinomia esadecimale é la seguente:

$a_{n-1}\ 16^{n-1}+a_{n-2}\ 16^{n-2}+.....+a_1\ 16^1+a_0$

la forma polinomia rende possibile la conversione da esadecimali a decimale.
es.

$(C34BD)_{16}=$
$C\ 16^4+3\ 16^3+4\ 16^2+B\ 16^1+D\ 16^0=$
$12\ 16^4+3\ 16^3+4\ 16^2+11\ 16^1+13\ 16^0=$
$(799933)_{10}$

Per la conversione da decimale a esadecimale si procede con il metodo delle divisioni successive
es.

		resto
64570	/ 16 = 4035	$10=A=a_0$
4035	/ 16 = 252	$3=3=a_1$
252	/ 16 = 15	$12=C=a_2$
15	/ 16 = 0	$15=F=a_3$

segue che $(64570)_{10}=(FC3A)_{16}$

Per le conversioni tra numeri esadecimali e numeri binari è necessario osservare le analogie fra forma polinomia binaria e forma polinomia esadecimale

$....+a_7\ 2^7+a_6\ 2^6+.....+a_1\ 2^1+a_0\ 2^0=$
$....+(a_7\ 2^3+a_6\ 2^2+a_5\ 2^1+a_4\ 2^0)\ 2^4+(a_3\ 2^3+a_2\ 2^2+a_1\ 2^1+a_0\ 2^0)\ 2^0=$
$....+\ (b_1)_{16}\ 16^1+\ (b_0)_{16}\ 16_0$

se ne deduce che ogni coefficiente esadecimale può essere espresso con 4 bit.

Per le conversioni tra numeri esadecimali e numeri binari si tenga presente la seguente tabella :

Esadecimale	Binario	decimale
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Esempio di conversione da esadecimale a binario in cui ogni cifra viene convertita nel corrispondente numero binario a quattro cifre:

$$\begin{array}{cccccc} (& A & D & 5 & F &)_{16} \\ & 1010 & 1101 & 0101 & 1111 & \end{array}$$

segue che $(AD5F)_{16}= (1010110101011111)_2$

Esempio di conversione binario-esadecimale ottenuta utilizzando le rispettive corrispondenze:

$$(0110/0111/1001/1101)_2$$
$$\begin{array}{cccc} 6 & 7 & 9 & D \end{array}$$

segue che$(0110011110011101)_2=(679D)_{16}$

Per la conversione della parte frazionaria, analogamente a quanto visto per la parte intera, si raggruppano quattro addendi alla volta partendo dalla cifra più significativa e spostandosi verso destra.

es.

$$(0.10110111101011)_2=(0.B7AC)_{16}$$

3.5 Numeri con Base Mista

Qualunque numero intero *x* può essere espresso, secondo il sistema di numerazione posizionale , nella forma polinomia:

$$x = a_{n-1} \times \xi + a_{n-2} \times \omega + a_{n-3} \times \sigma + \ldots\ldots + a_1 \times \beta + a_0 \times \alpha \quad (1)$$

con $\left(a_{n-1}a_{n-2}a_{n-3}.....a_1a_0 \right)_{\xi\omega\sigma.....\beta\alpha}$ dove $a_{n-1}, a_{n-2}, a_{n-3}, ...a_1, a_0$ sono *n* cifre distinte le une dalle altre, mentre $\alpha, \beta, \sigma, ..., \omega, \xi$ sono basi numeriche che, in teoria, possono essere un qualsiasi intero maggiore di uno e…….forse non e' necessario…….. con $\alpha \le \beta \le \sigma \le ....\le \omega \le \xi$……..
I valori che le cifre $a_{n-1}, a_{n-2}, a_{n-3}, ...a_1, a_0$ possono assumere nell'espressione (1) dipendono dalla base a cui esse sono associate ossia:

Cifra	Intervallo dei valori assunti
a_0	$0 \le a_0 < \alpha$
a_1	$0 \le a_1 < \beta$
....	....
a_{n-3}	$0 \le a_{n-3} < \sigma$
a_{n-2}	$0 \le a_{n-2} < \omega$
a_{n-1}	$0 \le a_{n-1} < \xi$

Data una sequenza di cifre, secondo il sistema di numerazione posizionale, si ha che:

- un **peso** è associato ad ogni posizione nella sequenza;
- ogni cifra indica quante volte deve essere considerato il peso corrispondente alla posizione nella quale si trova la cifra stessa.

Considerando la relazione (1) il peso associato alla cifra più a destra è α^0, il peso associato alla cifra successiva a sinistra, per una base mista, è $\alpha^1\beta^0$ e così via, ovvero, costruendo la seguente tabella, possiamo riepilogare tutti i pesi associati alle cifre della forma polinomia:

Peso	Cifra relativa alla base ad essa associata	Forma polinomia
α^0	$(a_0)_\alpha$	$a_0 \times \alpha^0$
$\alpha^1 \times \beta^0$	$(a_1)_\beta$	$a_1 \times \alpha^1 \times \beta^0$
....	....	....
$\alpha^1 \times \beta^1 \times .... \times \sigma^0$	$(a_{n-3})_\sigma$	$a_{n-3} \times (\alpha^1 \times \beta^1 \times .... \times \sigma^0)$

$\alpha^1 \times \beta^1 \times .... \times \sigma^1 \times \omega^0$	$(a_{n-2})_{\omega}$	$a_{n-2} \times (\alpha^1 \times \beta^1 \times .... \times \sigma^1 \times \omega^0)$
$\alpha^1 \times \beta^1 \times .... \times \sigma^1 \times \omega^1 \times \xi^0$	$(a_{n-1})_{\xi}$	$a_{n-1} \times (\alpha^1 \times \beta^1 \times .... \times \sigma^1 \times \omega^1 \times \xi^0)$

Concludendo possiamo scrivere la forma polinomia, in base mista, di un numero intero nel seguente modo:

$$x = a_{n-1} \times (\alpha^1 \times \beta^1 \times ... \times \sigma^1 \times \omega^1 \times \xi^0) + a_{n-2} \times (\alpha^1 \times \beta^1 \times ... \times \sigma^1 \times \omega^0) + a_{n-3} \times (\alpha^1 \times \beta^1 \times ... \times \sigma^0) + ...$$
$$..... + a_1 \times (\alpha^1 \times \beta^0) + a_0 \times \alpha^0$$

Dalla precedente espressione possiamo notare come ciascuna base, in ciascun addendo del numero x, sia al più di grado uno.

Esempio:

Dato il numero x, la sua espressione in forma polinomiale nelle basi 2, 3 e 5 risulta:

$$x = a_2 \times \gamma + a_1 \times \beta + a_0 \times \alpha$$

dove $\alpha = 2$, $\beta = 3$, $\gamma = 5$.

Sostituendo a ciascuna base il suo peso, relativamente alla sua posizione, si ottiene:

$$x = a_2 \times 2^1 \times 3^1 \times 5^0 + a_1 \times 2^1 \times 3^0 + a_0 \times 2^0$$

ossia

$$x = a_2 \times 6 + a_1 \times 2 + a_0 \times 1$$

Per determinare i coefficienti $a_{n-1}, a_{n-2}, a_{n-3}, ... a_1, a_0$ della forma polinomia, in base mista, di un qualunque intero x si può costruire la tabella delle possibili combinazioni fra le basi presenti nel polinomio.

Esempio

Dato il numero x, la sua espressione in forma polinomiale nelle basi 2, 3 e 5 risulta:

$$x = a_2 \times \gamma + a_1 \times \beta + a_0 \times \alpha$$

dove $\alpha = 2$, $\beta = 3$, $\gamma = 5$.

I valori che a_0, a_1, a_2, possono assumere sono:

- per a_0 sono $\{0,1\}$ (essendo a_0 associato ad α);
- per a_1 sono $\{0,1,2\}$ (essendo a_1 associato a β);
- per a_2 sono $\{0,1,2,3\}$ (essendo a_2 associato a γ).

La tabella delle possibili combinazioni di cifre fra queste basi risulta:

Numero	$(a_0)_2$	$(a_1)_3$	$(a_2)_4$
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	2	0
5	1	2	0
6	0	0	1
7	1	0	1
8	0	1	1
9	1	1	1
10	0	2	1
11	1	2	1
12	0	0	2
13	1	0	2
14	0	1	2
15	1	1	2
16	0	2	2
17	1	2	2
18	0	0	3
19	1	0	3
20	0	1	3
21	1	1	3
22	0	2	3
23	1	2	3

Ad esempio, in corrispondenza di $(a_0)_{\mathbf{2}}$=1, $(a_1)_{\mathbf{3}}$ =1, $(a_2)_{\mathbf{4}}$=3, utilizzando la relazione (2), si ottiene il numero 21:

$$x = 3\times\left(2^1\times 3^1\times 4^0\right)+1\times\left(2^1\times 3^0\right)+1\times 2^0 = 3\times 6+1\times 2+1 = 21$$

Osservazione

Se la forma polinomia, in base mista, di un intero è del tipo:

$$x = a_4\times\gamma+a_3\times\gamma+a_2\times\gamma+a_1\times\beta+a_0\times\alpha$$

utilizzando la formula (2) risulta:

$$x = a_4\times\left(\alpha^1\times\beta^1\times\gamma^2\times\gamma^0\right)+a_3\times\left(\alpha^1\times\beta^1\times\gamma^1\times\gamma^0\right)+a_2\times\left(\alpha^1\times\beta^1\times\gamma^0\right)+a_1\times\left(\alpha^1\times\beta^0\right)+a_0\times\alpha^0$$

ossia il grado di una base può essere maggiore di 1, se questa è presente in almeno tre addendi consecutivi nella rappresentazione polinomiale. In questo caso, infatti, γ risulta di secondo grado.

Inoltre dobbiamo aggiungere che all'interno dell'espressione polinomiale una stessa base può essere presente in più termini, ma con peso diverso:

$$x = a_5\times\beta+a_4\times\gamma+a_3\times\alpha+a_2\times\gamma+a_1\times\beta+a_0\times\alpha$$

ovvero, utilizzando l'espressione (2), si ha

$$x = a_5\times\left(\alpha^1\times\beta^1\times\alpha^1\times\gamma^1\times\beta^0\right)+a_4\times\left(\alpha^1\times\beta^1\times\alpha^1\times\gamma^0\right)+a_3\times\left(\alpha^1\times\beta^1\times\gamma^1\times\alpha^0\right)+a_2\times\left(\alpha^1\times\beta^1\times\gamma^0\right)+a_1\times\left(\alpha^1\times\beta^0\right)+a_0\times\alpha^0$$

da cui si deduce che per a_0, α ha peso 1, mentre per a_3 α ha peso $\alpha^1\times\beta^1\times\gamma^1$; per a_1 β ha peso α^1 mentre per a_5 βla stessa base ha peso $\alpha^2\times\beta^1\times\gamma^1$.

In modo analogo, un numero frazionario y (normalizzato a zero) può essere rappresentato nella forma polinomia come:

$$y = a_0\times\alpha+a_{-1}\times\beta+a_{-2}\times\gamma+a_{-3}\times\sigma+.....+a_{-m}\times\xi \qquad (3)$$

dove a_0 = 0, con $\left(\ 0.a_{-1}a_{-2}a_{-3}.....a_{-m}\ \right)_{\alpha\beta\gamma\sigma.....\xi}$.

Qualunque sia il numero rappresentato, la cifra più a sinistra è la cifra più significativa del numero, la cifra più a destra è la cifra meno significativa del numero. Una cifra è più significativa di un'altra cifra se la sua posizione nel numero è più a sinistra della posizione dell'altra cifra.

Dalla relazione (3), si possono ricavare i pesi associati alle varie cifre, come indicato nella seguente tabella:

Peso	**Cifra relativa alla base ad essa associata**	**Forma polinomia**
α^0	$(a_0)_\alpha$	$a_0 \times \alpha^0$
$\alpha^{-1} \times \beta^0$	$(a_{-1})_\beta$	$a_{-1} \times \alpha^{-1} \times \beta^0$
$\alpha^{-1} \times \beta^{-1} \times \gamma^0$	$(a_{-2})_\gamma$	$a_{-2} \times (\alpha^{-1} \times \beta^{-1} \times \gamma^0)$
$\alpha^{-1} \times \beta^{-1} \times \gamma^{-1} \times \sigma^0$	$(a_{-3})_\sigma$	$a_{-3} \times (\alpha^{-1} \times \beta^{-1} \times \gamma^{-1} \times \sigma^0)$
.....	.....	.....
$\alpha^{-1} \times \beta^{-1} \times \gamma^{-1} \times \sigma^{-1} \times .... \times \xi^0$	$(a_{-m})_\xi$	$a_{-m} \times (\alpha^{-1} \times \beta^{-1} \times \gamma^{-1} \times \sigma^{-1} \times .... \times \xi^0)$

In questo modo posso scrivere la (3) come:

$$y = a_0 \times \alpha^0 + a_{-1} \times (\alpha^{-1} \times \beta^0) + a_{-2} \times (\alpha^{-1} \times \beta^{-1} \times \gamma^0) + a_{-3} \times (\alpha^{-1} \times \beta^{-1} \times \gamma^{-1} \times \sigma^0) + ... + a_{-m} \times (\alpha^{-1} \times \beta^{-1} \times \gamma^{-1} \times \sigma^{-1} \times .... \times \xi^0)$$

Il Metodo delle divisioni successive come quello delle moltiplicazioni successive deve tenere conto della sequenza delle basi utilizzate pertanto si deve applicare una conversione diversa per ogni coefficiente.

3.6 Forma complemento

Si definisce forma complemento a b di un numero intero N in base b con k cifre quel numero che sommato a N da':

$$C_b(N) = b^k - N$$

O equivalentemente $N + C_b(N) = b^k$

Si ricordi che il valore b^k è dato, qualunque sia la base *b*, dal primo simbolo della base dopo lo zero e quindi in base 10 e 2 si ha che e' un 1 seguito da *k* cifre uguali a zero

$10^k \rightarrow 10^4 = 10000$

Es. k=4 b^k $2^k \rightarrow 2^k = 10000$

Es. $C_{10}(562)$ Dunque $b^k = 1000$

$$C_{10}(562) = 1000 - 562 = 438$$

Es. Prendendo un numero binario: $N = (101)_2$

$$b^k = 1000 \qquad C_2(101) = 1000 - 101 = 11$$

3.6.1 Complemento a b-1

Si definisce complemento a $b-1$ di un numero intero *N*, rappresentato in base b con k cifre, il numero $C_{b-1}(N)$ in base *b* tale che:

$$C_{b-1}(N) = b^k - 1 - N$$

$$N + C_{b-1}(N) = b^k - 1$$

Si nota che essendo b^k-N-1=C_b(N)-1, otteniamo $C_{b-1}(N) = C_b(N) - 1$

Si ricordi che il valore $b^k - 1$ è rappresentato, qualunque sia la base *b*, da *k* cifre tutte uguali a $b-1$ cioe' all'ultimo simbolo della base. Per esempio in base 10 a una serie di 9 ed in base 2 ad una serie di 1. In tali basi si parla pertanto di complemento a 9 ed ad 1 rispettivamente.

Esempio. Nel sistema decimale:

C_9(35722)=100000-35722-1=64277
Infatti N(=35722)+C_9(N)(=64277)=99999

Esempio. Nel sistema binario:
C1(10011)=100000-10011-1=11111-100110=1100
Infatti C1(N)+N=11111

3.6.2 Proprietà e relazioni

Il complemento gode delle seguenti proprietà:

1) $C_b(N) = C_{b-1}(N) + 1$

Dim.: $N + C_{b-1}(N) = b^k - 1$

$C_{b-1}(N) + 1 = b^k - N$

dalla definizione di complemento si ottiene: $C_{b-1}(N) + 1 = C_b(N)$

2) $C_b(C_b(N)) = N$

Dim.: ricordando che $C_b(N)=b^k-N$, cioè $N=b^k-C_b(N)$
Si ottiene, sostituendo ad N tale valore nella seconda parte di questa stessa espressione,

$N=b^k-C_b(b^k-C_b(N))$

La seconda parte di questa espressione non è altro che $C_b(C_b(N))$ sviluppato.

3) $C_{b-1}(C_{b-1}(N)) = N$

Dim.: sapendo che $N=b^k-C_{b-1}(N)-1$
Adesso, sostituendo tale valore di N nella seconda parte della stessa aquazione, otteniamo

$N=b^k-C_{b-1}(b^k-C_{b-1}-1)-1$

Che non è altro se non la definizione di $C_{b-1}(C_{b-1}(N))$

3.6.3 Complementi Veloci

Per calcolare il complemento a 1 e a 2 di un numero binario si può ricorrere ai seguenti procedimenti:

Complemento a 1 veloce:
-Il complemento a 1 di un numero binario si ottiene trasformando le cifre zero in uno e viceversa (che equivale ad eseguire un'operazione **not**).

Tale operazione deriva direttamente dalla proprietà 1); infatti:

$C_{b-1}(N) = (b^k - 1) - N$

Ricordiamoci che b^k-1 equivale, in base 2, ad un numero costituito da k zeri; inoltre si nota come l'operazione di sottrazione dalla cifra 1 equivalga ad un'operazione di **not**

Es. Complemento a 1 veloce

$k = 6 \quad N = 101001$

$2^6 - 1 = 111111$

$$\begin{array}{l} 111111- \\ \underline{101001=} \\ 010110 \rightarrow \text{quindi } C_{b-1}(N)_2 = not(N)_2 \end{array}$$

L'operazione di sottrazione fra 1 e 0 e fra 1 e 1 sono equivalenti all'applicazione dell'operatore not.

Complemento a 2 veloce

Per calcolare più velocemente il complemento a 2 si puo' utilizzare il seguente algoritmo:

Tutto ciò si può riassumere dicendo che il complemento a 2 di un binario si ottiene riscrivendo così come sono tutti i bit del numero a partire da quello meno significativo fino al primo bit uguale a 1, compreso, e eseguendo l'operazione **not** su tutti gli altri.

Es.: complemento a 2 veloce

N=10001101; C_1(N)=01110010;

C_2(N)=C_1(N)+1=01110011 che rispecchia il caso in cui l'ultima cifra di (N) sia uno.

Es.:complemento a 2 veloce

N=01101100; C_1(N)=10010011;

C_2(N)=C_1(N)+1=10010100 che rispecchia il caso in cui l'ultima cifra di (N) sia zero (in questo caso l' algoritmo di pagina 38 è stato percorso tre volte).

3.6.4 Sottrazioni con il complemento

Per calcolare la differenza $X - Y$ tra due numeri in una qualunque base *b*, dove *X* è un numero di *k* cifre si possono eseguire le seguenti operazioni:

$$X-Y=X-Y+b^k-b^k=$$

$$=X+b^k-Y-b^k=$$
$$=X+C_b(Y)-b^k$$

Per esempio si supponga di effettuare la sottrazione tra numeri decimali $742-593=149$
Questa differenza può essere calcolata come:

$$742-593+1000=$$
$$742+(1000-593)=$$
$$742+407=$$
$$1149=X-Y+b^k$$

Infatti per ottenere 149 (cioè X-Y) basta togliere b^k(cioè 10^3=1000). Questo e' la cifra 1 che 'avanza' a sinistra.

Ricordiamo che bk espresso in qualunque base è seguito da k zeri: perciò tale differenza si riduce al non considerare il traboccamento.

con questo procedimento ho trasformato una sottrazione in una somma tra *X* e il complemento di *Y*. Ciò permette di effettuare una differenza senza avere alcuna operazione di prestito.

$$\text{Es.}(X)_2=11001100 \qquad (Y)_2=11000010$$
$$C_2(Y)=00111110$$

$$11001100+$$
$$\underline{00111110=}$$
$$\text{traboccamento che verrà trascurato}\rightarrow 1|00001010|$$

Trascurare la cifra più significativa equivale a sottrarre al risultato ottenuto b^k in questo caso 2^8,cioè $(100000000)_2$

ESERCIZI CAPITOLO 3

1)Si dica a quale numero decimale corrisponde il seguente numero binario:011011010.1010 (soluzione : $2^7+2^6+2^4+2^3+2+2^{-1}+2^{-3}=128+64+16+8+2+0.5+0.125=218.625$

2) Si convertino da decimali a binari i seguenti numeri:
(a) 532.14 (b) 893.97 calcolando gli eventuali errori commessi, fermandosi a sei cifre significative nella parte frazionale
(Soluzione (a) : - parte intera (metodo delle divisioni successive)

	resto
532:2=266	0
266:2=133	0
133:2=66	1
66:2=33	0
33:2=16	1
16:2=8	0
8:2=4	0
4:2=2	0
2:2=1	0
1:2=0	1

-parte frazionaria (metodo delle moltiplicazioni sucessive)

	parte intera
0.14x2=0.28	0
0.28x2=0.56	0
0.56x2=1.12	1
1.12x2=0.24	0
0.24x2=0.48	0

$(532.14)_{10}$ è circa $(1000010100.00100)_2$
L'errore assoluto è : $E_a=V_v-V_r= 532.14-532.125=0.015$
L'errore relativo è : $E_r= E_a / V_v= 0.015/532.14=2.81x10^{-5}$
L'errore percentuale è : $E_{\%}= E_r x\ 100=2.81x10^{-3}$
L'errore massimo è : $2^{-k}=2^{-5}=0.03125$

Soluzione (b) – parte intera

Resto

893:2=446 1
446:2=223 0
223:2=111 1
111:2=55 1
55:2=27 1
27:2=13 1
13:2=6 1
6:2=3 0
3:2=1 1
1:2=0 1

-parte frazionaria

0.97x2=1.94 1
0.94x2=1.88 1
0.88x2=1.76 1
0.76x2=1.52 1
0.52x2=1.04 1

$(893.97)_{10}$ è $(1101111101.11111)_2$

E_a=893.97-893.96875=0.00125

E_r=0.00125/893.97=$1.398x10^{-6}$

$E_{\%}$=$1.398x10^{-4}$

3) Calcolare A+B con A=$(10111011)_2$ e B=$(00110110)_2$

		verifica decimale
(Soluzione:	10111011 +	187 +
	00110110	54
	11110000	241

4) Calcolare A-B con A=$(1000001)_2$ e B=$(0000101)_2$

		verifica decimale
(Soluzione:	1000001 -	65-
	0000101	5
	0111100	60

5) Calcolare AxB con A=$(1001010)_2$ e B=$(1001)_2$

		verifca decimale
(Soluzione	1001010 x 1001	74 x 9 = 666
	1001010	
	1001010 - - -	
	1010011010	

6) Calcolare A/B con A=$(1100)_2$ e B=$(101)_2$

(soluzione

```
1100.00 | 101          verifica decimale
101     | 10.01        12:5=2.4
----    |
0010.00
 1.01
 0.11
```

Infatti la coversione della parte decimale in binario è :

0.4x2=0.8	0
0.8x2=1.6	1
0.6x2=1.2	1

7) Si converta da decimale a ottale il numero 426

(Soluzione

	resto
426:8=53	2
53:8=6	5
6:8=0	6

$(426)_{10}$ corrisponde a: $(652)_8$

8) A quale numero ottale corrisponde $(101110110)_2$?

(Soluzione : raggruppando il numero a cifre di 3 si ottiene

101|110|110

5 6 6

$(101110110)_2$ è $(566)_8$

9) Si convertano i seguenti numeri esadecimali in decimali e binari :

(a) $(AF5)_{16}$ (b) $(FE8)_{16}$

(Soluzione: (a) $Ax16^2+Fx16+5=10x16^2+15x16+5=(2805)_{10}$

$AF5=(101011110101)_2$

(b) $Fx16^2+Ex16+8=15x16^2+14x16+8=(4072)_{10}$

$FE8=(111111101000)_2$

4 Rappresentazione dei Dati

4.1 Codifica dell'informazione

Tutte le informazioni inserite in un elaboratore, o emesse dall'elaboratore stesso sono rappresentate nel cosiddetto *alfabeto esterno,* i cui caratteri generalmente sono le 26 lettere (maiuscole e minuscole) dell'alfabeto inglese, le 10 cifre decimali, i vari segni di interpunzione (tra i quali è compreso anche lo spazio), i simboli matematici, e i vari caratteri di controllo.
All'interno dell'elaboratore però i dati devono essere rappresentati con l'*alfabeto interno* che è quello binario, i cui unici caratteri sono 0 e 1. Questa necessità è data dal fatto che tutte le componenti di un elaboratore (transistor, nastri magnetici, etc...) possono trovarsi soltanto in due stati, corrispondenti ovviamente alle cifre 0 e 1.
E' chiaro allora che ogni volta che vengono immesse delle informazioni all'interno dell'elaboratore queste debbano essere *codificate*, cioè tradotte dall'alfabeto esterno a quello interno. Ugualmente le informazioni emesse dall'elaboratore devono essere *decodificate*. All'interno di un elaboratore possono essere utilizzati anche codici diversi (purché ovviamente siano utilizzati in parti diverse dell'elaboratore stesso).
Le codifiche più usate per i dati di tipo alfanumerico, sui quali cioè non è possibile eseguire operazioni numeriche, sono l'EBCDIC (Extended Binary Code Decimal Interchange Code) e soprattutto l'ASCII (American Standard Code for Information Interchange). In entrambe le codifiche ogni carattere richiede un byte per essere rappresentato; il primo semibyte (nibble) è detto *zone,* il secondo *digit*.

4.2 Codifica EBCDIC

Con l'EBCDIC è possibile rappresentare lettere, numeri e altri simboli.Per ciascuno di essi sono utilizzati 8 bit, cioè un byte, perciò le possibili combinazioni sarebbero 256.
In verità non tutte le combinazioni vengono utilizzate. Tale codice è molto diffuso negli elaboratori IBM.

				zone	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				digit	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	0	0		NUL	DLE	DS		SP	&										0
0	0	0	1		SOH	DC1	SOS				/		a	j			A	J		1
0	0	1	0		STX	DC2	FS	SYN					b	k	s		B	K	S	2
0	0	1	1		ETX	TM							c	l	t		C	L	T	3
0	1	0	0		PF	RES	BYP	PN					d	m	u		D	M	U	4
0	1	0	1		HT	NL	LF	RS					e	n	v		E	N	V	5
0	1	1	0		LC	BS	ETB	UC					f	o	w		F	O	W	6
0	1	1	1		DEL	IL	ESC	EOT					g	p	x		G	P	X	7
1	0	0	0			CAN							h	q	y		H	Q	Y	8
1	0	0	1			EM							i	r	z		I	R	Z	9
1	0	1	0		SMM	CC	SM		¢	!		:								
1	0	1	1		VT	CU1	CU2	CU3	·	$	,	#								
1	1	0	0		FF	IFS		DC4	<	*	%	@								
1	1	0	1		CR	IGS	ENQ	NAK	(	)	–	'								
1	1	1	0		SO	IRS	ACK		+	;	>	=								
1	1	1	1		SI	IUS	BEL	SUB	|	¬	?	"								

4.3 Codifica ASCII

Per quanto riguarda la codifica ASCII è bene precisare che originariamente era una codifica a 7 bit e solo in un secondo momento è stata forzata a diventare a 8 bit ponendo il bit più significativo uguale a zero. Attualmente esiste una versione (codice ASCII *esteso*) che sfrutta anche l'ottavo bit per un totale di 256 codifiche.
Di seguito è riportata la tabella ASCII nella versione precedente a quella estesa

				Z	0	0	0	0	0	0	0	0
				O	0	0	0	0	1	1	1	1
				N	0	0	1	1	0	0	1	1
				digit	0	1	0	1	0	1	0	1
0	0	0	0		NUL	DLE	SP	0	@	P		p
0	0	0	1		SOH	DC1	!	1	A	Q	a	q
0	0	1	0		STX	DC2	“	2	B	R	b	r
0	0	1	1		ETX	DC3	#	3	C	S	c	s
0	1	0	0		EOT	DC4	$	4	D	T	d	t

0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	‘	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O		o	DEL

Es. La codifica ASCII di 72 è 00110111|00110010.

**** Si noti che le codifiche ASCII e EBCDIC sono utilizzate per rappresentare informazioni di tipo alfanumerico. Anche le sequenze di cifre come “72” non sono dunque, come già detto, numeri sui quali poter effettuare operazioni, ma semplici informazioni espresse da numeri, come per esempio numeri telefonici. I numeri veri e propri si esprimono nelle forme già viste (valore assoluto con segno, rappresentazione complemento a 1 e a 2).****

4.4 Codifica BCD

Per rappresentare solo quantità di tipo numerico, c’è un’altra codifica: la BCD (Binary Coded Decimal). Essa trasforma ogni cifra decimale nel corrispondente numero binario. Poiché la cifra decimale più grande (9) corrisponde a $(1001)_2$ per rappresentare una cifra serviranno 4 bit.

Per esempio la codifica BCD di 56 è
0101 0110 (la codifica binaria di 56 invece è 111000).

Tabella BCD

Cifra Decimale	Codifica BCD
0	0000
1	0001
2	0010

3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Es: somme con riporto

Vogliamo sommare 72 e 35;

72 in BCD 0111 0011 ; 35 in BCD 0011 0101;
7 2 3 5

0111 0011 +
0011 0101 =
1010 1000

↘ però questo numero non esiste in BCD, perciò il risultato verrà scritto, usando un linguaggio corretto, come

0001 0000 1000
1 0 8 che fornisce il corretto risultato.

4.5 Confronto fra le varie rappresentazioni

Possiamo già fare un confronto tra le varie rappresentazioni di numeri in termini di spazio occupato, ossia di byte, per registrare la stessa informazione.

Esempio: $(4567)_{10}$ Vediamo quanto spazio occupa questa informazione numerica nelle varie rappresentazioni:

ASCII	BCD	Binario	Ottale	Esadecimale
32 bit	16 bit	13 cifre	5 cifre	4 cifre
(1 byte per cifra)	(4 bit per cifra)			

4.6 Codifica dei Numeri Interi

Per codificare i numeri possono essere usati più codici binari a seconda del tipo di informazione numerica (numeri frazionari,interi...). La rappresentazione binaria in virgola fissa (o fixed point) rappresenta ciascun numero assumendo una posizione prefissata del punto di separazione e utilizza una quantità prefissata di memoria. In particolare la rappresentazione in virgola fissa può essere usata per rappresentare i numeri interi. Sono possibili più codifiche che differiscono per il criterio usato nella rappresentazione dei numeri negativi:

-Forma valore assoluto con segno
-Forma complemento a 2
-Forma complemento a 1

4.6.1 Forma valore assoluto con segno

Consiste nel rappresentare il segno e il valore assoluto (o modulo) del numero intero separatamente. Qualunque sia il numero n di bit usati, il bit più a sinistra, cioè la cifra più significativa, è utilizzata per rappresentare il segno (0 per un numero positivo, 1 per un numero negativo); i restanti $n-1$ bit rappresentano la codifica binaria del modulo.

Es. per n=8 si ha:

Numero	Segno	Modulo
+13	0	0001101
–13	1	0001101

Con questo tipo di rappresentazione il rango dei numeri rappresentabili con n bit è:

$$-2^{n-1}+1,-2^{n-1}+2,\ldots,-0,+0,\ldots,2^{n-1}-2,2^{n-1}-1$$

Perciò si ha un rango di $\pm(2^{n-1}-1)$

Per esempio Codifica con 8 bit

$$-(2^7-1),\ldots,-0,+0,\ldots,2^7-1$$
$$-127,\ldots,-0,+0,\ldots,127$$

Riportiamo una tabella semplificativa

127	0111 1111
	:
2	0000 0010
1	0000 0001
+0	0000 0000
-0	1000 0000
-1	1000 0001
-2	1000 0010
	:
-127	1111 1111

Quindi è possibile rappresentare 255 numeri binari diversi (non 256 perché ho una doppia rappresentazione dello zero:
0 0000000 e 1 0000000).

4.6.2 Rappresentazione complemento a 2

La rappresentazione nella forma complemento a 2 è la più usata per rappresentare i numeri durante la fase di elaborazione. Essa consiste nel rappresentare i numeri positivi nella tradizionale forma binaria e ogni numero negativo $-X$ per mezzo del complemento a 2 del corrispondente numero positivo *+X*. In pratica, con la codifica nella forma complemento a 2, si ha che l'usuale valore posizionale è associato a ciascuna cifra binaria con l'esclusione del bit più significativo al quale è associato un valore negativo; in pratica inoltre, il bit più significativo è indice del segno del numero (0=+, 1=--), anche se riveste un altro significato numerico.

Ad esempio se si utilizzano 8 bit si ha che i pesi associati alle cifre del numero binario sono:

-128 64 32 16 8 4 2 1.

Il rango dei numeri rappresentabili con n bit è:

$$-2^{n-1},-2^{n-1}+1,\ldots,0,\ldots,2^{n-1}-2,2^{n-1}-1.$$

cioè rango $-2^{n-1} \leq N \leq 2^{n-1}-1$

in questo caso si andrebbe da -128 a 127.

Es. n=8

Dovendo codificare in forma complemento a 2 il numero decimale 53 si esegue la conversione in forma binaria ottenendo così 110101. Poiché si hanno sei cifre e si vuole eseguire una rappresentazione a 8 bit è necessario aggiungere due zeri a sinistra: 00110101.

Se invece si vuole rappresentare -53 si esegue il complemento a 2 di 00110101, ottenendo così

11001011

Adesso è possibile rappresentare 256 numeri,compreso lo zero;vediamolo in tabella

127	0111 1111
2	0000 0010
1	0000 0001
0	0000 0000
-1	1111 1111
-2	1111 1110
-3	1111 1100
-128	1000 0000

Attenzione a non confondere la rappresentazione in forma complemento a 2 con il complemento a 2 di un numero.

Oggi comunque si lavora perlopiù con 16 bit(-2^{15};2^{15}-1) oppure con 32 bit, raggiungendo un rango di circa 2 miliardi.

Facendo operazioni di somma e differenza tra due numeri in forma complemento a 2 ci si può trovare di fronte a dei riporti scomodi (anche perchè il bit di segno è trattato allo stesso modo degli altri bit).

- Il riporto sul bit di segno viene chiamato *carry;* mi cambia quindi il segno del risultato.
- Il riporto al di fuori del bit di segno viene chiamato *overflow;* il risultato è quindi al di fuori del rango dei numeri permessi da questo tipo di rappresentazione.

Si possono presentare le seguenti situazioni:

1) no carry no overflow

00100010+
<u>11010101=</u>
11110111

In questo caso non si ha né overflow né carry, e l'operazione risulta corretta.

2) sia carry che overflow

$$\begin{array}{r} 00100010+ \\ \underline{11110101=} \\ 1\ 00010111 \end{array}$$

In questo caso si hanno sia carry che overflow. Il risultato dell'operazione è corretto in quanto si è ottenuto (in decimale): $34-11=23$. La cifra più significativa del risultato può essere trascurata perché è il 10^k della definizione di complemento $X-Y=X+C_2(N)-\underline{2^k}$ (guarda paragrafo 3.7.4)

3) carry senza overflow (ERRORE)

$$\begin{array}{l} 00100010+ \\ \underline{01110101=} \\ 10010111 \end{array}$$

Si ha un carry senza overflow. L'elaboratore segnalerà la presenza del carry in quanto fornirà un risultato dell'operazione non corretto. Infatti: 34+117= 151 che è fuori dalla dinamica dei numeri permessi. L'elaboratore legge comunque il risultato. In questo caso risulterebbe: -128+16+4+2+1= --105
Qualunque combinazione degli 8 bit fornisce un "possibile risultato". Infatti questa **rappresentazione** si dice **chiusa,** ossia aggiungendo 1 al numero più grande, cioè 127, si riottiene –128,che è il più piccolo, come se le 256 combinazioni formassero un unico collegamento chiuso.
Ritornando all'esempio occorre notare che se però successivamente si ha un'altra operazione (si parla di operazioni in cascata) con un solo overflow allora si ritorna al caso precedente e quindi il risultato delle due operazioni è corretto. Per esempio volendo eseguire l'operazione $103+90-70=123$ all'interno dell'elaboratore si ha:

$$\begin{array}{l} 01100111+ \\ \underline{01011010=} \\ 11000001 \end{array}$$

Il risultato dell'operazione $103+90$ sarebbe dunque un numero negativo per la presenza del carry. Sommando però -70 il risultato finale dell'operazione è corretto. L'overflow, infatti, ha compensato il traboccamento.

$$\begin{array}{r} 11000001+ \\ \underline{10111010=} \\ \text{cifra trascurata} \rightarrow 1\ 01111011=(123)_{10} \end{array}$$

4) overflow senza carry (ERRORE)

$$\begin{array}{r} 10010111+ \\ \underline{10100001=} \\ 1\ 00111000 \end{array}$$

Si ha un overflow senza carry, o anche **overflow reale,** quindi il risultato non è corretto.Infatti $-105-95=-200$. L'elaboratore dunque segnalerà la presenza dell'overflow. Se però successivamente si ha un'operazione con un solo carry il risultato tra le due operazioni è corretto come nel caso precedente.

4.6.3 Rappresentazione complemento a 1

Questo tipo di rappresentazione è simile a quella nella forma complemento a 2 ma è molto meno diffusa. Nella forma complemento a 1 un numero negativo è rappresentato come il complemento a 1 del corrispondente positivo; in questo modo il peso negativo associato alla cifra più significativa corrisponde a $-2^{n-1}+1$.

Ad esempio se si usano 8 bit, i valori posizionali per la forma complemento a 1 sono:

-127 64 32 16 8 4 2 1.

Il rango dei numeri rappresentabili con n bit è:

$$-2^{n-1}+1,-2^{n-1}+2,\ldots,0,\ldots,2^{n-1}-2,2^{n-1}-1$$

rango $\pm(2^{n-1}-1)$, nel caso in cui n=8 avremo [-127;+127]

Adesso per cui si rappresentano 255 numeri; rispetto alla rappresentazione complemento a 2 si perde infatti la combinazione 1111 1111 che prima significava -1, adesso invece rappresenta lo zero.

127	0111 1111
1	0000 0001
+0	0000 0000
-0	1111 1111
-1	1111 1110
-127	1000 0000

Anche nel caso della forma complemento a 1 il bit di segno è trattato nello stesso modo degli altri bit quindi si possono verificare carry e overflow.

Esempio.

$$\begin{array}{r} 11001010+ \\ \underline{11000101=} \\ 1\ 10001111 \end{array}$$

In questo caso si hanno un carry e un overflow. Per ottenere il risultato corretto bisogna sommare alla cifra meno significativa il valore dell'overflow cioè 10001111 + 1 = 10010000. Infatti si ottiene $-53-58=-111$. Questo metodo si chiama *wrap around carry.*

Infatti $N\text{-}M$ corrisponde, utilizzando la rappresentazione complemento a 1, a

$$N\text{-}M= N+C_{b-1}(M)-b^k+1$$

ma $$M=C_{b-1}(M)-b^k+1$$

Da ciò deriva $$N-C_{b-1}(M)=N-M+b^k-1$$

Ma siccome a noi interessa $N\text{-}M$ occorrerà togliere dal risultato ottenuto b^k e aggiungere un'unità: avremo così raggiunto il corretto risultato.

Esempio: dimostriamola stessa proprietà lavorando con un numero speciale, ossia lo zero meno(1111 1111) e aggiungendovi un'unità

1111 1111+
0000 0001=
1 0000 0000 Ma che cosa dice la regola?

Di togliere l' overflow e di aggiungerlo alla cifra meno significativa !
Otterremo così, infatti ,il corretto risultato (0000 0001)
Ciò è stato possibile perché si sono realizzati contemporaneamente sia un. carry che un overflow

4.7 Bit, Nibble, Bytes e Word

Un bit rappresenta una cifra binaria. Il bit è però un'unità di informazione troppo piccola per poter essere elaborata in modo efficiente. I bit pertanto sono trattati secondo i gruppi che seguono:

1 nibble = 4 bit
1 byte = 8 bit
1 word = 16 bit
1 doubleword = 32 bit
1 Kilobyte = 2^{10} byte=1024 byte=8196 bit
1 Megabyte = 2^{20} byte=1048576 byte~8 milioni bit
1 Gigabyte = 2^{30} byte~ 1miliardo byte~8 miliardi bit
1 Terabyte = 2^{40} byte~10^{12} byte~2^{43} bit

La dinamica dei tipi del Turbo Pascal

Tipo	Intervallo	Bit occupati
Nibble	0—15	4
Byte	0—255	8
Shortint	-128—127	8
Word	0—65535	16
Doubleword	0—4294967295	32
Integer	-32768—32767	16
Longint	-2147483648—2147483647	32

4.7 Codifica in Virgola Mobile

4.7.1 Richiami sulla rappresentazione in virgola fissa

Per la codifica di un numero possiamo usare più codici binari a seconda del tipo e della natura dell'informazione numerica considerata.
La rappresentazione fixed point o a virgola fissa si basa sull'impiego di un numero fisso di cifre per la rappresentazione della parte intera e per la parte frazionaria, secondo il seguente schema:

Il limite più evidente di una tale rappresentazione, è quello di poter esprimere solo un rango limitato di valori che può risultare spesso insufficiente per le applicazioni correnti; in particolare avendo a disposizione K cifre per la parte intera, la dinamica esprimibile è $\pm 2^K$ estremi esclusi, mentre la precisione, avendo m cifre per la parte frazionaria, è $\leq 2^{-m}$.
Per questo motivo si estende questa codifica ad una più capace da un punto di vista della dinamica; questo nuovo tipo di rappresentazione prende appunto il nome *di codifica in virgola mobile.*

4.7.2 Codifica in virgola mobile

Nei casi in cui si renda necessario un più ampio rango dei numeri rappresentabili, la codifica in virgola mobile risulta una delle soluzione più usate; è questo il caso di problemi tecnici e scientifici in cui il risultato di una operazione, o gli stessi addendi, possono comportare una accuratezza tale da dover

richiedere l'uso di una dinamica e di una precisione ben al di sopra di quella possibile con la rappresentazione in virgola fissa.
La rappresentazione di un numero in virgola mobile o floating point consiste nel prodotto di due parti: il fattore scala, che è una potenza del 10 (nel caso generale) e della parte frazionaria o significativa tale che, moltiplicata per il fattore di scala fornisce il numero desiderato.
Esempio:

$$1.15 \times 10^{3} = 1150$$

E' immediato notare come questo tipo di rappresentazione non sia unico; lo stesso numero 1150 può avere infatti diverse rappresentazioni:

$$1150 = 1.15 \times 10^{3} = 0.115 \times 10^{4} = 0.00115 \times 10^{6} \text{ .... ecc.}$$

E' utile notare come un numero qualsiasi, non normalizzato, lo può diventare giocando sull'esponente; ad esempio:

$$0.02 \times 10^{-7} \Rightarrow 0.2 \times 10^{-8}$$

virgola a sinistra di un posto +1 <- (+1)
virgola a destra di un posto -1 -> (-1)

Per quanto riguarda le diverse forme di rappresentazione in virgola mobile, l'IEEE (The Institute of Electrical and Electronics Engineerings) ha definito alcune ' forme ' di rappresentazione che sono usate da tutti gli elaboratori; esse differiscono tra loro solo per il numero di bit riservati alla rappresentazione della mantissa e della caratteristica e hanno tutte la seguente forma generale:

S	CARATTERISTICA	MANTISSA

dove:
- s è un bit riservato segno che può essere 0 (numero positivo) o 1 (numero negativo)
- la caratteristica è l'esponente (del due nel caso binario) del numero rappresentato
- la mantissa è la parte frazionaria (ciò che è dopo lo 0.) che, moltiplicata per 2 elevato alla caratteristica, fornisce il numero cercato.

Di queste rappresentazioni ne vedremo alcune.

4.7.3 Rappresentazione reale corto

Usa 32 bit: 23 per la mantissa, 8 di caratteristica e 1 di segno.

1	8	23

i bit sono numerati da 0 a 31 a partire da quello meno significativo cioè quello più a destra.

4.7.4 Rappresentazione reale lungo

Usa 64 bit: 52 per la mantissa, 11 di caratteristica e 1 di segno.

1	11	52

4.7.5 Rappresentazione ' versione a 16 bit '

Usa 16 bit: 10 per la mantissa, 5 di caratteristica e 1 di segno.

1	5	10

Vediamo adesso come si intende per 0 nel calcolatore, infatti ci sono due possibili ' zero ':
- quello derivante da una operazione, come ad esempio 0.0×10^{3} ; questo però può succedere ad esempio per problemi legati alla cancellazione numerica derivante dalla sottrazione di due numeri molto vicini tra loro
- 0.0×10^{0} che è l'effettiva rappresentazione dello zero per il calcolatore.

4.7.6 Rappresentazione della caratteristica

La forma della caratteristica è quella dell'eccesso a t cioè :

$$\mathbf{C = E + t = E + (2^{K - 1} - 1)}$$

dove :

- C = caratteristica
- E = esponente in binario
- K = bit a disposizione per la rappresentazione della caratteristica

La forma di eccesso a t è usato per evitare l'uso di un bit di segno per la rappresentazione dell'esponente.

Consideriamo infatti K = 8 (come nella forma di reale corto):

$t = 2^{8 - 1} - 1 = 127$, quindi $C = E + 127$

C	0	1	2	3	127 ..	.	255
E	**-127***	-126....	....	...	0 ...		128

* per convenzione al posto di questo valore si considera lo 0 effettivo.

Dinamica

Considerando sempre la rappresentazione *reale corto* si ha una dinamica pari a :

$$\pm 2^{t+1} = \pm 2^{128}$$

Precisione
Per quanto riguarda la precisione, definita come il numero più piccolo rappresentabile, per la convenzione del secondo zero risulta:

$$\pm 2^{-t+1} = \pm 2^{-126}$$

Errore
Quando converto un numero in una rappresentazione in virgola mobile, se questo non è esattamente esprimibile come somma di potenze del 2, commetto sempre una imprecisione valutabile in termini di errore assoluto, relativo e percentuale.

4.7.7 Errore assoluto

$$E_a = V_v - V_R$$

dove:

- V_v = è il valore vero do rappresentare
- V_R = valore rappresentato in virgola mobile

4.7.8 Errore relativo

$$E_R = \frac{E_A}{V_V} = \frac{V_V - V_R}{V_V}$$

4.7.9 Errore percentuale

$$E_P = \frac{E_A}{V_V} \cdot 100 = \frac{V_V - V_R}{V_V} \cdot 100$$

Osservazione
si faccia particolare attenzione al fatto che l'errore assoluto dipende dall'ordine di grandezza dei dati considerati, mentre l'errore relativo e quello percentuale no.
Per esempio: scrivere 999 invece di 1000 o 999000 invece di 1000000 comporta, a parità di errore relativo (0.1) e percentuale (0.1 %), un errore assoluto di 1 nel primo caso e 1000 nel secondo.

4.7.10 Esempio di calcolo dell'errore nella conversione di un numero

1) Volendo rappresentare il numero 3002 in virgola mobile procediamo nel seguente modo:
- lo convertiamo in binario con il metodo delle divisioni successive e lo normalizziamo a zero, ottenendo :

$$(3002)_2 = 0.101110111010 * 10^{1100}$$

L'errore nasce dalla finitezza della mantissa: se nel nostro caso disponiamo di 10 bit per la sua rappresentazione vengono tagliate le ultime 2 cifre, ottenendo la rappresentazione del numero 3000. Abbiamo, in particolare:

$$E_a = 3002 - 3000 = 2$$

$$E_R = 2 / 3002 = 0.666* 10^{-3}$$

$$E_P = 0.0666 \%$$

2) Supponiamo di voler rappresentare adesso 3422641

$$(3422641)_2 = 0.1101000011\underline{100110110001} * 10^{10110}$$

avendo per esempio 10 cifre di mantissa commettiamo un errore di 2481 cioè rappresentiamo il numero 3420160, abbiamo
quindi :

$$E_a = 3422641 - 3420160 = 2481$$

$$E_R = 2481 / 3422641 = 0.7* 10^{-4}$$

$$E_P = 0.007 \%$$

concludiamo quindi che al crescere del numero di cifre troncate l'errore assoluto cresce e in maniera non lineare.
Nota:
rappresentando il numero con una caratteristica limitata (a 10 cifre nel nostro caso), la parte che viene troncata (quella sottolineata in precedenza) è proprio l'errore assoluto che si commette (infatti abbiamo troncato 100110110001 = 2481 = E_a).
3) Supponiamo di voler convertire il numero 35.96
in questo caso, per quanto detto sopra, introdurremo un errore dovuto alla finitezza della caratteristica; infatti:

$$(35.96)_2 = 0.1000111111 * 10^{110}$$

dove ci siamo interrotti alla 10 ° cifra della caratteristica.

Operando in questo modo, ciò che abbiamo rappresentato non è il numero 35.96 ma

$$(0.1000111110 * 10^{110})_{10} = 35.9375$$

Abbiamo così :

$$E_a = 35.96 - 35.9375 = 0.0225$$

$$E_R = 0.0225 / 35.96 = 0.62 * 10^{-3}$$

$$E_P = 0.062\ \%$$

4.8 Operazioni fra Numeri in Virgola Mobile

4.8.1 Somma fra Numeri in Virgola Mobile

Supponiamo di dover sommare due numeri già espressi in virgola mobile, il procedimento si compone delle seguenti fasi:
1) bisogna rendere uguali le due caratteristiche degli addendi, traslando opportunamente la mantissa del numero con caratteristica più piccola
2) la caratteristica della somma è uguale alla caratteristica comune (dopo aver eseguito l'operazione del punto 1).
3) La mantissa della somma è uguale alla somma rinormalizzata (a zero o a uno a seconda dei casi) delle mantisse dei numeri.

4.8.2 Esempio: somma in virgola mobile

Supponiamo di dover sommare tra loro i seguenti numeri : (sempre nella forma a 16 bit)

$$-N_1 = 0.511 * 10^3$$

$$-N_2 = 0.153 * 10^4$$

Abbiamo :

$$(N_1 = 0.511 * 10^3 = 511)_2 = 0.111111111 * 10^{1001}$$

$$(N_2 = 0.153 * 10^4 = 1530)_2 = 0.1011111101 * 10^{1011}$$

Usando 5 cifre di caratteristica l'eccesso risulta $t = 2^{5-1} - 1 = 15$ che in binario si scrive 1111.
La rappresentazione in virgola mobile dei due numeri è la seguente:

$$511 = 0\text{-}\ 11000\ \text{-}1111111110$$

$$1530 = 0\ \text{-}11010\ \text{-}1011111101$$

La prima operazione da eseguire è portare la caratteristica minore , che è quella di 511, al valore di quella più grande; avendo aumentato la caratteristica, devo eseguire uno shift della mantissa, ottenendo quindi :

$$511 = 0\ \text{-}11010\ \text{-}\ 0011111111$$

$$1530 = 0\ -11010\ -\ 1011111101$$

La somma avrà :
- come segno 0 (è la somma tra due numeri positivi)
- come caratteristica quella comune ai due numeri (11010)
- come mantissa la somma delle mantisse (0.1111111100) .

511 + 1530 = 2041 = 0 -11010 -1111111100

Nel caso che la somma delle mantisse avesse un riporto oltre la virgola andrebbe rinormalizzata a zero ottenendo il troncamento della cifra meno significativa (vedi esempio).
Nell'esempio otteniamo (la caratteristica vera del numero ottenuto come somma è quella del risultato in virgola mobile diminuito dell'eccesso) :

$$(0.11111111 * 10^{1011})_{10} = 2040$$

Abbiamo cioè :

$$E_a = 2041 - 2040 = 1$$

$$E_R = 1 / 2041 = 0.48 * 10^{-4}$$

$$E_P = 0.0048\ \%$$

4.8.3 Esempio (rinormalizzazione della mantissa della somma)

Sommare i seguenti numeri:

$$x_1 = 3.5$$

$$x_2 = 1.25$$

Abbiamo:
$(x_1 = 3.5)_2 = 0.111 * 10^{10001}$ (già espresso in forma di eccesso a t)
$(x_2 = 1.25)_2 = 0.101 * 10^{10000}$ (già espresso in forma di eccesso a t)
dobbiamo adesso modificare la caratteristica minore, cioè quella di x_2 ; avremo cioè :

$$(x_2 = 1.25)_2 = 0.0101 * 10^{10001}$$

La somma, al solito, avrà :
- come segno 0 (essendo la somma di due numeri positivi),
- come caratteristica quella comune ai due addendi
- come mantissa la somma della due mantisse :

$$\text{Mantissa}(x_1) + \text{Mantissa}(x_2) \Rightarrow \begin{array}{r} 0.1110 + \\ 0.0101 = \\ \hline 1.0011 \end{array}$$

Come avevamo già anticipato, la mantissa della somma deve essere normalizzata:
Abbiamo cioè :

$$\text{Mantissa}(x_1 + x_2) = 0.10011$$

Questo implica l'aumento di 1 della caratteristica, che diventa:

$$C(x_1 + x_2) = 10^{10010}$$

4.8.4 Esempio : errore dovuto a troncamento della mantissa

Durante l'esecuzione della somma, possiamo avere un altro caso particolare: quello in cui si ha un troncamento della mantissa della somma, dovuto ad una rinormalizzazione della medesima; questo ovviamente introduce un errore.
Si dato:

$$x_1 = 3.50390625$$

$$x_2 = 1.25$$

Abbiamo:
$(x_1 = 3.50390625)_2 = 0.1110000001 * 10^{10001}$ (già espresso in forma di eccesso a t)
$(x_2 = 1.25)_2 = 0.101 * 10^{10000}$ (già espresso in forma di eccesso a t)
dobbiamo adesso modificare la caratteristica minore, cioè quella di x_2 ; avremo cioè :

$$(x_2 = 1.25)_2 = 0.0101 * 10^{10001}$$

La somma, al solito, avrà :
- come segno 0 (essendo la somma di due numeri positivi),
- come caratteristica quella comune ai due addendi
- come mantissa la somma della due mantisse :

$$\text{Mantissa}(x_1) + \text{Mantissa}(x_2) \Rightarrow \begin{array}{r} 0.1110000001 + \\ 0.0101000000 = \\ \hline 1.0011000001 \end{array}$$

la normalizzazione provoca un troncamento nella mantissa :

$$0.1001100000 \quad \boxed{1}$$

Abbiamo quindi :

$$x_1 + x_2 = 4.75390625$$

$$x_1 \oplus x_2 = 4.75$$

$$E_a = 4.75390625 - 4.75 = 0.00390625$$

$$E_R = 0.00390625 / 4.75390625 = 0.00082$$

$$E_P = 0.082\ \%$$

4.8.5 Prodotto fra Numeri in Virgola Mobile

Supponiamo adesso di dover moltiplicare due numeri già espressi in virgola mobile normalizzati a zero ed espressi nella forma con eccesso a t; il procedimento di prodotto si compone delle seguenti fasi:
1) Segno : segue le usuali regole di calcolo; nel dettaglio :

0 x 0 = 0
0 x 1 = 1
1 x 0 = 1
1 x 1 = 0

2) Caratteristica: la caratteristica del prodotto di due numeri in virgola mobile è pari alla somma delle caratteristiche dei due numeri (entrambi espressi nella forma di eccesso a t) diminuita di t.
Cioè:

$$\text{Caratteristica } (x_1 * x_2) = \text{Car} (x_1) + \text{Car} (x_2) - t$$

questo è dovuto al fatto che il risultato dell'espressione appena scritta sarebbe in realtà :
$E_1 + t + E_2 + t = (E_1 + E_2) + 2t$, e per riportarlo nella forma ' canonica $E + t$ ', deve essere appunto diminuito di t medesimo.
3) Mantissa: la mantissa del prodotto è il prodotto binario usuale delle mantisse eventualmente rinormalizzata.

4.8.6 Esempio : prodotto in virgola mobile

Si voglia eseguire il prodotto dei due numeri seguenti:

$$x_1 = 2.25$$

$$x_2 = 1.5$$

$(x_1 = 2.25)_2 = 10.01 = 0.1001 * 10^{10}$ = (nella forma con eccesso a t) = $0.1001 * 10^{10001}$
$(x_2 = 1.5)_2 = 1.1 = 0.11 * 10^{1}$ = (nella forma con eccesso a t) = $0.11 * 10^{1000}$
Avremo:
1) segno :

$$0 \text{ x } 0 \Rightarrow 0$$

2) Caratteristica:

Caratteristica $(x_1 * x_2) = Car(x_1) + Car(x_2) - t \Rightarrow$

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

C1 = E1 + t

C2 = E2 + t

C1+C2 = E1 + E2 + 2t

Ecco perche' e' necessario detrarre t.

3) Mantissa: prodotto binario tra le due mantisse:

```
      0 . 1 0 0 1 *
          0 . 1 1  =
  --------------------
        0 1 0 0 1
      0 1 0 0 1 -
 --------------------
  0 . 0 1 1 0 1 1
```

<u>Osservazione importante:</u>

- Bisogna eseguire il prodotto con gli 0. in colonna per il corretto calcolo del posizionamento finale della virgola
- Nella esecuzione del calcolo si risparmia tempo nel considerare come secondo fattore quello a mantissa più corta
- Moltiplicando il primo fattore per 1 si riscrive il numero stesso e si scala di un posto a capo, moltiplicando per zero si scala semplicemente di un posto.

Adesso rinormaliziamo la mantissa del prodotto , shiftando di uno la caratteristica; il prodotto sarà allora:

0 -1 0 0 0 1- 1 1 0 1 1 0 0 0 0 0

Per convertire il risultato è necessario diminuire di t la caratteristica del prodotto, il ' vero ' numero rappresentato è dunque:

0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0

Infatti :

$$(0 - 0 0 0 1 0 - 1 1 0 1 1 0 0 0 0 0)_{10} = (0.11011 * 10^{10})_{10} = 3.375$$

----------fine prima parte ----------