di GIORGIO OBER (GIOBE2000)

l'angolo di Mr A.KEER

(parte quindicesima)

PROGETTARE con le PORTE LOGICHE

Sottrazione Logica

Il progetto digitale deve provvedere, con sofisticate macchine combinatorie, al supporto di tutte le operazioni aritmetiche; in questa puntata ci occupiamo dei dispositivi chiamati a fare la sottrazione dei numeri binari.

La speciale rassegna delle ultime due puntate ci ha portato a conoscere i componenti logici adatti a fare la somma di numeri binari; vedremo ora come gli stessi possano essere coinvolti anche per la differenza aritmetica, passando per la definizione del concetto di *Complemento* di un numero.

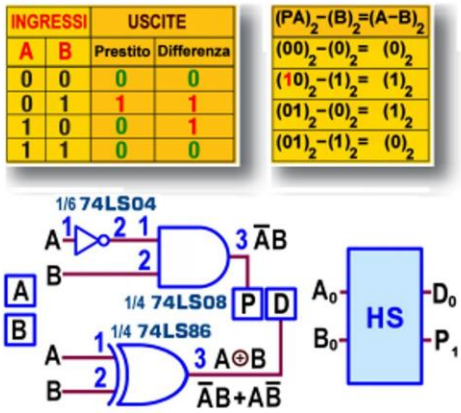
Sottrattore Parallelo

Tutti noi abbiamo imparato che la sottrazione di numeri (decimali) può richiedere un "*Prestito* di 10" se il *Minuendo* è minore del *Sottraendo*; naturalmente questo meccanismo rimane inalterato anche quando i numeri trattati appartengono al sistema di numerazione binario, fermo restando che il valore della base (chiesta "in *Prestito*") è ora 2, cioè $(10)_2$.

Come abbiamo fatto per i sommatori è conveniente creare una struttura paragonabile alla coppia **HA** (*Half Adder*) e **FA** (*Full Adder*) in grado di sostenere direttamente (con l'uso di sole porte logiche) l'operazione di *Sottrazione* aritmetica; possiamo cominciare con il circuito chiamato ad esprimere quella di 2 numeri di 1 bit senza *Prestito*, ottenuto a partire dalla sua *Tabella di Verità*.

Il progetto di un *Sottrattore* ad 1 bit deve dunque prevedere 2 funzioni, una per la *Differenza*, D_0, ed una per il *Prestito*, P_1, spesso definito *Borrow*, all'inglese); la *Figura 1* mostra la sequenza dei valori assunti dalle 2 funzioni in corrispondenza delle 4 possibili combinazioni delle 2 variabili d'ingresso A_0 e B_0, rappresentanti i numeri binari (a 1 bit) da sottrarre tra loro.

La realizzazione del nostro progetto (noto come *Mezzo Sottrattore*, *Half Subtractor*, **HS**) è dunque molto semplice; esso costituisce la base di partenza per creare sottrattori di numeri espressi da una quantità di bit grande a piacere; non è difficile riconoscere la funzione **XOR** in quella destinata a realizzare la *Differenza* D_0 (lo stesso operatore utilizzato dal *mezzo sommatore* **HA** per assicurare la Somma S_0) mentre per tener conto del *Prestito* (P_1) è necessaria la funzione **AND** con l'ingresso associato al *Minuendo* sottoposto ad inversione logica prima di essere fornito alla porta.



INGRESSI		USCITE	
A	B	Prestito	Differenza
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

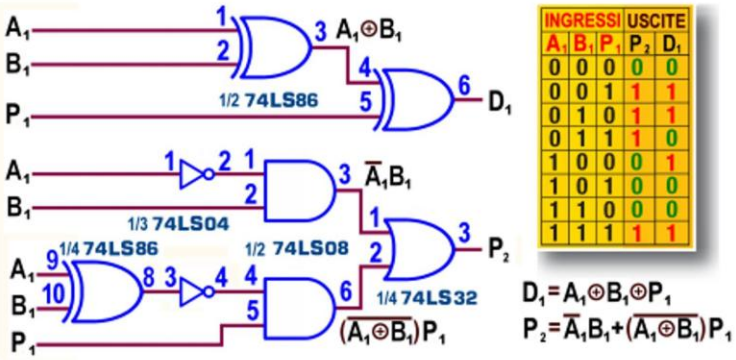
$(PA)_2-(B)_2=(A-B)_2$
$(00)_2-(0)_2=(0)_2$
$(10)_2-(1)_2=(1)_2$
$(01)_2-(0)_2=(1)_2$
$(01)_2-(1)_2=(0)_2$

Figura 1 - 1-bit (Half) Subtractor: Tabelle di verità (in logica positiva) e schemi

Va quindi sottolineato che gli operandi coinvolti non possono essere scambiati tra loro, cioè devono mantenere la loro valenza di *Minuendo* (A_0) e di *Sottraendo* (B_0) per garantire la funzionalità del progetto nei termini proposti dalla *Figura 1*.

Come per i sommatori la semplice struttura appena realizzata non è sufficiente per operare la *Differenza* delle cifre binarie di peso superiore a quello dei bit meno significativi (A_0, B_0) dei 2 numeri: a partire da quelli di peso 1 (A_1, B_1) si dovrà tener conto anche dell'eventuale *Prestito* richiesto dalla *Differenza* delle 2 cifre immediatamente precedenti.

Il nuovo progetto dovrà garantire ancora 2 funzioni d'uscita (*Prestito* P_2 e *Differenza* D_1) ma a partire da 3 ingressi; la *Figura 2* mostra la Tabella di verità, le formule di progetto da essa dedotte e le relative reti logiche.



INGRESSI			USCITE	
A_1	B_1	P_1	P_2	D_1
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Figura 2 - 1-bit (Full) Subtractor: Tabelle di verità (in logica positiva) e schemi

L'insieme dei 2 circuiti è noto come *Sottrattore Completo* (*Full Subtractor*, **FS**); le formule sono ottenute rielaborando quelle canoniche, costitute da 4 mintermini; poiché si vede chiaramente che entrambe affidano una parte del lavoro alla **XOR** di A_1 (*Minuendo*) con B_1 (*Sottraendo*) sembra logico utilizzarne una sola, condivisa dai 2 circuiti.

La *Figura 3* mostra il **FS** dopo questo accorpamento.

Ridisegnando il circuito è facile verificare (*Figura 4*) che il **FS** è sostanzialmente costituito da 2 **HS** in cascata con in più una **OR**; la *Figura 5* mostra la sintesi a blocchi di questa situazione.

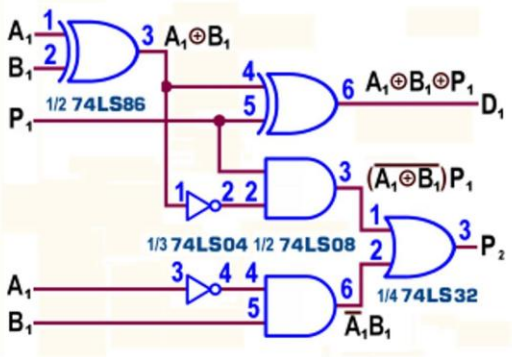


Figura 3 - 1-bit (Full) Subtractor: schema 1

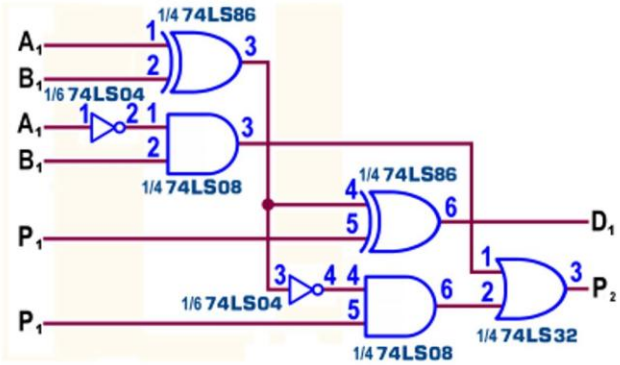


Figura 4 - 1-bit (Full) Subtractor: schema 2

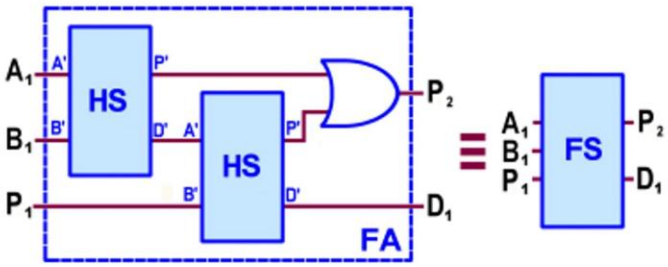


Figura 5 - 1-bit (Full) Subtractor: schema 3

In stretta analogia con le strutture fondamentali dei sommatori (**HA** e **FA**) il calcolo della *Differenza* di numeri ad **n** bit può essere gestita affidando la coppia dei rispettivi bit meno significativi (A_0, B_0) ad un **HS** e le rimanenti **n-1** coppie ad altrettanti **FS**; la *Figura 6* mostra un Sottrattore a 4 bit (**n**=4) chiamato a togliere dal *Minuendo* A=$(1100)_2$=$(12)_{10}$ il *Sottraendo* B=$(1001)_2$=$(9)_{10}$; il risultato, $(0011)_2$=$(3)_{10}$, è facilmente verificabile operando uno dopo l'altro i singoli bit di uguale peso, a cominciare da quelli meno significativi, in accordo con le regole della sottrazione aritmetica, coinvolgendo l'eventuale *Prestito* generato dalla sottrazione dei bit precedenti.

Da notare che il *Prestito* P_4 del **FS** più significativo non viene coinvolto per stabilire il valore del risultato ma può essere inteso come segno della differenza: se (come nell'esempio ora proposto) il *Minuendo* A è maggiore o uguale al *Sottraendo* B (A>=B) esso è lasciato a **0** ed è associato al segno + (risultato positivo).

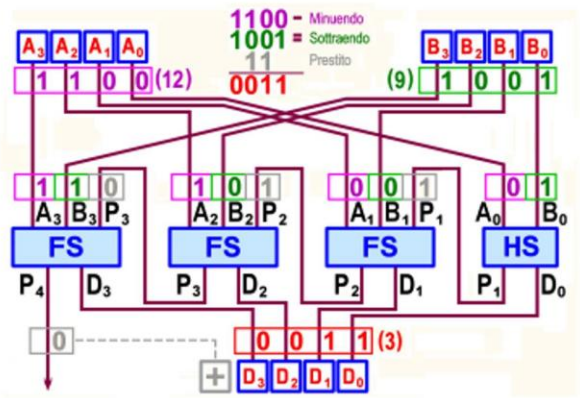


Figura 6 - 4-bit Ripple Borrow Subtractor con A>=B

Lo schema di questo ***Sottrattore Parallelo*** mette in evidenza il percorso ad *ondulazione* (***Ripple***) affidato alle linee che gestiscono la propagazione del *Prestito* (***Borrow***) dalla posizione meno significativa fino a quella di maggior peso; per analogia con i sommatori, questa struttura si può definire ***Ripple Borrow Subtractor*** (sottrattori "*a propagazione del Prestito*").

Concetto di Complemento alla Base

Prima di proseguire è necessario fissare le idee sul modo corretto di intendere i Numeri Binari: di fatto una sequenza di n bit può essere interpretata come numero "con segno" o "senza segno", ma si tratta di una convenzione, dato che il dispositivo chiamato ad utilizzarla tratterà gli 1 e gli 0 sempre e solo come livelli logici alternativi.

Il diverso modo di interpretare la sequenza dipende esclusivamente dal *contesto* e spetta al progettista (o al programmatore, nel caso dei processori) imporre la strategia circuitale (o di programma) ad esso necessaria; la *Sottrazione aritmetica* è proprio uno dei casi tipici in cui bisogna tener conto della convenzione affidata agli operandi.

Il primo problema consiste nello stabilire in che modo possa essere aggiunto il segno + o il segno - ad una cifra binaria: è consuetudine affidarne il valore al suo bit più significativo, fissato a 0 per indicare un *numero positivo* e a 1 per indicare un *numero negativo*.

Finora tutte le stringhe binarie proposte sono state intese come *numeri interi senza segno*: in esse non ci siamo mai curati di specificare il fantomatico segno +, ma la cosa risulta legittimata dal fatto che gli eventuali 0 non significativi posti davanti alla sequenza dei bit effettivi (rappresentanti il *modulo*) del numero lo rendono implicitamente *positivo*.

Da questo punto di vista, per esempio, i numeri $(01100)_2$ e $(11100)_2$ sono rispettivamente uguali a $(12)_{10}$ e a $(28)_{10}$ (se interpretati senza segno) o a $(+12)_{10}$ e a $(-12)_{10}$ (se ritenuti con segno); è evidente la assoluta necessità di dichiarare esplicitamente a priori la modalità con cui dovranno essere trattate le informazioni numeriche, per evitare ogni possibile confusione.

Questa premessa dà corpo alla prima definizione di numeri con segno, detta "*Modulo e Segno*" per il fatto che al valore del codice binario puro a n bit che lo rappresenta (detto *modulo*) viene anteposto un ulteriore bit a 0 o a 1, per il segno; in coerenza con le premesse una stringa di 4 bit esprime 8 valori positivi (da +0 a +7) e 8 valori negativi (da -0 a -7), i secondi ricavati dai primi semplicemente cambiando il bit di segno, come si vede nella colonna a sinistra della *Figura 7*.

Si nota la presenza di 2 rappresentazioni per lo zero, il che rende poco pratica questa convenzione, anche dal punto di vista aritmetico, richiedendo di gestire separatamente i bit di

segno e quelli di modulo (trattati come numeri senza segno) e imponendo talvolta la correzione del risultato, in particolare proprio con la sottrazione.

Per questa ragione è sorta l'esigenza di rappresentare in altri modi i numeri con segno, utilizzando tecniche legate al valore del *Complemento* (*alla base* o *alla base -1*) del numero.

NUMERI con segno MODULO e SEGNO				
+7	0	1	1	1
+6	0	1	1	0
+5	0	1	0	1
+4	0	1	0	0
+3	0	0	1	1
+2	0	0	1	0
+1	0	0	0	1
+0	0	0	0	0
-0	1	0	0	0
-1	1	0	0	1
-2	1	0	1	0
-3	1	0	1	1
-4	1	1	0	0
-5	1	1	0	1
-6	1	1	1	0
-7	1	1	1	1

NUMERI con segno COMPLEMENTO a 1				
+7	0	1	1	1
+6	0	1	1	0
+5	0	1	0	1
+4	0	1	0	0
+3	0	0	1	1
+2	0	0	1	0
+1	0	0	0	1
+0	0	0	0	0
-0	1	1	1	1
-1	1	1	1	0
-2	1	1	0	1
-3	1	1	0	0
-4	1	0	1	1
-5	1	0	1	0
-6	1	0	0	1
-7	1	0	0	0

NUMERI con segno COMPLEMENTO a 2				
+7	0	1	1	1
+6	0	1	1	0
+5	0	1	0	1
+4	0	1	0	0
+3	0	0	1	1
+2	0	0	1	0
+1	0	0	0	1
+0	0	0	0	0
-1	1	1	1	1
-2	1	1	1	0
-3	1	1	0	1
-4	1	1	0	0
-5	1	0	1	1
-6	1	0	1	0
-7	1	0	0	1
-8	1	0	0	0

Figura 7 - Rappresentazione dei Numeri binari con Segno

Il *Complemento alla Base meno uno* di un numero si ottiene sottraendolo da un altro con la stessa quantità di cifre, tutte uguali al simbolo di valore più grande del sistema di numerazione di riferimento; con quello decimale (*Base*=10) si parla di "*Complemento a 9*" (per esempio 10345 lo è di 89654=99999-10345) mentre con il sistema di numerazione binario (*Base*=2) si parla di "*Complemento a 1*" (per esempio 10110 lo è di 01001=11111-10110).

Dalla colonna centrale di *Figura 7* si nota che il modulo dei *numeri positivi* a 4 bit rimane uguale a quello dei numeri senza segno mentre quello dei *numeri negativi* è il risultato della sottrazione del modulo positivo da 111; per esempio, per $(1010)_2=(-5)_{10}$, è pari a 010 (=111-101).

In sintesi: in binario il numero negativo si ottiene semplicemente complementando (invertendo) tutti i bit (compreso quello di segno) del corrispondente numero positivo; in "*Complemento a 1*" (come in "*Modulo e Segno*") si possono esprimere 8 valori positivi (da +0 a +7) e 8 valori negativi (da -0 a -7), ma si rilevano anche gli stessi limiti, sopratutto per il fatto di prevedere 2 rappresentazioni per lo zero.

La soluzione è dunque quella di affidare la costruzione dei numeri negativi alla tecnica del *Complemento alla Base*, che consiste nel sottrarre il numero A di **n** cifre dal valore $B^{\mathbf{n}}$; è facile dimostrare che esso si ottiene aggiungendo 1 a quello del *Complemento alla Base meno uno*.

Nel sistema di numerazione decimale (*Base*=10) si parla di "*Complemento a 10*" (per esempio quello di 10345 è 89655=100000-10345) mentre in quello binario (*Base*=2) si parla di "*Complemento a 2*"; la colonna a destra della *Figura 7* mostra che ora (tra i numeri con segno

a 4 bit) è previsto un solo valore per lo zero (ritenuto *positivo*, in un range da +7 a +0) e 8 valori negativi (da -1 a -8).

La completa assenza di ambiguità e la facilità di ottenerne i valori sommando 1 a quelli del corrispondente "*Complemento a 1*" (immediatamente disponibili con semplici inverter) rende questa tecnica affidabile e molto applicata nei dispositivi aritmetici.

Possiamo tornare ora al nostro ***Sottrattore Parallelo*** per mettere in evidenza una situazione che sarà comunque da tenere presente anche nei successivi progetti: se il *Sottraendo* B è maggiore del *Minuendo* A, il risultato della *Sottrazione* deve essere corretto; la *Figura 8* propone lo stesso circuito di *Figura 6*: il valore (senza segno) del *Minuendo* A=$(1001)_2$=$(9)_{10}$ è ora più piccolo di quello del *Sottraendo* B=$(1100)_2$=$(12)_{10}$; il risultato (ottenuto operando nello stesso modo di prima) indica il valore $(1101)_2$ che, interpretato senza segno = $(13)_{10}$, non ha alcun significato, ma che inteso in *Complemento a 2* è proprio (-3).

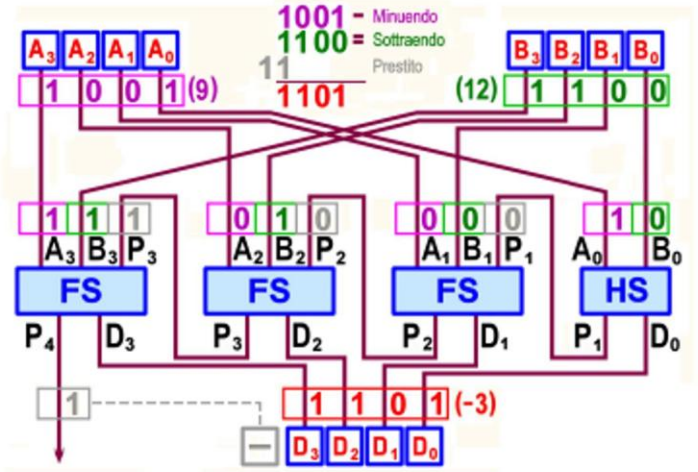


Figura 8 - 4-bit Ripple Borrow Subtractor con A<B

Per trasformare il risultato in *Modulo* e *Segno* (al fine di renderlo disponibile (per esempio) per il decoder di un digit a 7 segmenti) è necessario aggiungere una macchina combinatoria dedicata; la *Figura 9* mostra una possibile soluzione: il quadruplo multiplexer da 2 a una linea, 74LS157, lascerà passare inalterato il risultato (se A>=B, essendo già leggibile direttamente) oppure (se A<B) porterà in uscita una stringa di bit ottenuta da quella del *Sottrattore* dopo che su di essa è stato imposto il *Complemento a 2*.

La rete che si occupa di questo compito è facilmente progettabile con la consueta tecnica di minimizzazione delle mappe di Karnaugh ottenute a partire dalla Tabella di verità; le formule di progetto portano ad un circuito con sole 5 porte, 2 **OR** 74LS32 e 3 **OREX** 74LS86. La marcatura dei bit coinvolti, passo dopo passo, aiuta a capire la straordinaria efficacia di questo dispositivo e si presta alla verifica di tutte le altre possibili sequenze raccolte dalla colonna di destra della *Figura 7*.

E' interessante notare che la selezione tra le 2 possibilità è operata automaticamente a partire dal valore della linea di *Prestito* P_4 relativa al **FS** più significativo: coerentemente a quanto detto in precedenza, se il *Minuendo* A è maggiore o uguale al *Sottraendo* B non potrà esserci alcuna richiesta di *Prestito*, per cui tale linea sarà lasciata a **0**, non sarà necessario intervenire sul risultato e il suo segno sarà ritenuto positivo; viceversa, con P_4=**1** il risultato sarà negativo e il valore iniziale dovrà essere corretto (complementato a 2).

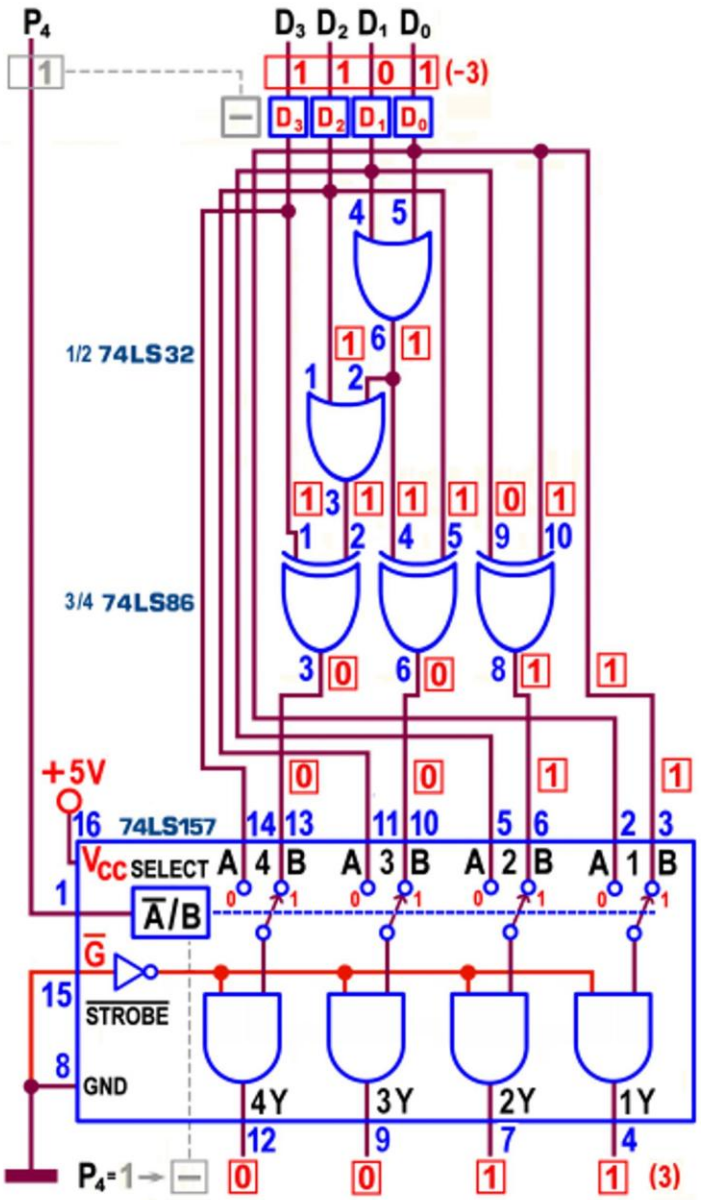


Figura 9 - 4-bit Ripple Borrow Subtractor con A<B (segue)

Per realizzare sottrattori grandi a piacere è naturalmente possibile collegare in cascata più moduli di tipo **FS**, per altro relativamente lenti: è facile capre che il risultato $D_n,...,D_0$ sarà attendibile solo dopo che l'informazione associata al *Prestito* avrà attraversato tutti i moduli presenti, così che il tempo necessario sarà **n** volte più grande di quello richiesto da ciascuno di loro.

Il mercato non prevede alcun integrato intrinsecamente *Sottrattore*, anche perchè (come vedremo tra poco) è possibile disporre nello stesso dispositivo sia della funzione di somma che di quella di differenza; esistono invece un paio di componenti in grado di operare l'operazione di *Complemento alla Base meno 1* su 4 bit.

Alla **serie TTL** appartiene la versione binaria, il **74LS87**, definito **4 bit True/Complement Zero/One Element** per sintetizzare tutti i servizi assicurati; la *Figura 10* mostra il suo *pin-out*.

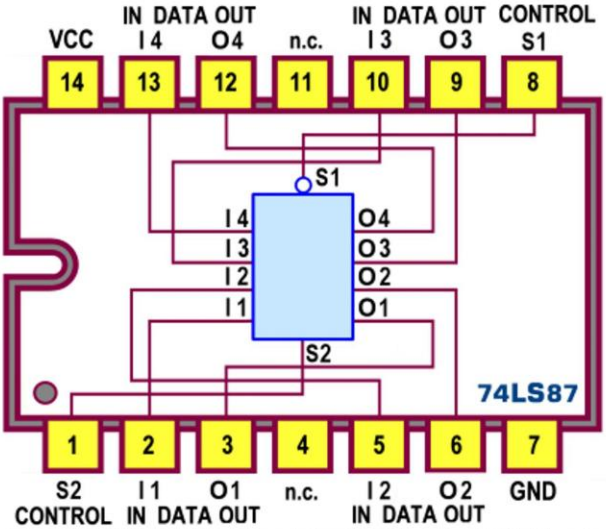


Figura 10 - 4-bit True/Complement Zero/One Element 74LS87: Pin-out

In realtà si tratta di un componente molto semplice in grado di forzare le sue 4 uscite in altrettanti stati predefiniti, in funzione del valore fissato sulle 2 linee di controllo S_1 e S_2.

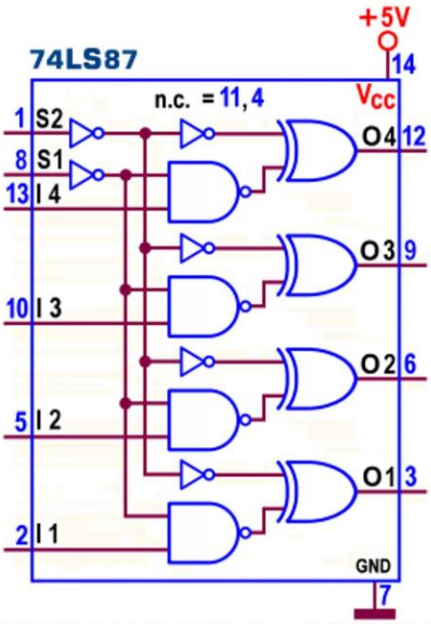


Figura 11 - 4-bit True/Complement Zero/One Element 74LS87: Schema funzionale

Lo *schema funzionale* di *Figura 11* aiuta a capire il suo modo di operare; quando l'ingresso di controllo S_1 è a **0** logico tiene aperte le porte **NAND**, consentendo il passaggio in uscita della quaterna di bit d'ingresso (*Data Inputs*), inalterata se S_2=**1** oppure in forma negata (cioè in *Complemento a 1*) se S_2=**0**; quando S_1=**1** logico le porte **NAND** sono chiuse e le loro uscite a **1** obbligano la rispettiva **OREX** a forzare tutte e 4 le uscite a **1** (se S_2=**0**) oppure a **0** (se S_2=**1**).

La *Figura 12* simula la situazione topica (quella con entrambi i controlli S_1 e S_2 a massa) mettendo in evidenza il ruolo decisivo delle porte **OREX** nella creazione in uscita del valore binario opposto ("complementato a 1") rispetto a quello fornito in ingresso.

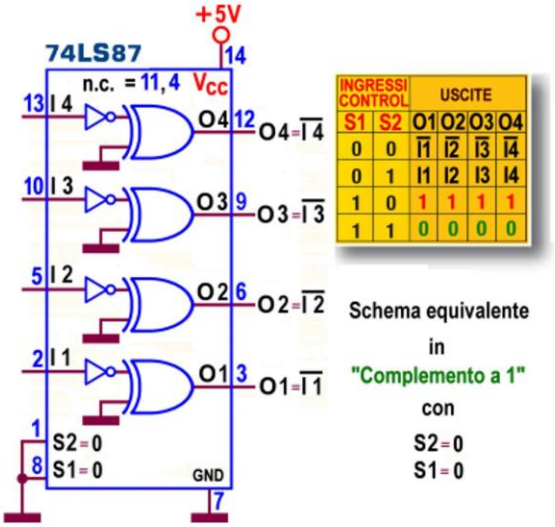


Figura 12 - 4-bit True/Complement Zero/One Element 74LS87: uso in Complemento a 1

Lo *schema pratico*, consigliato nella stesura dei progetti per la sua sintetica completezza è proposto in *Figura 13*.

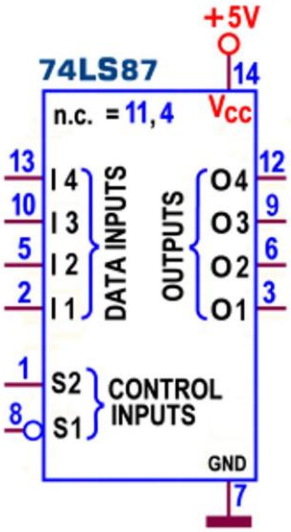


Figura 13 - 4-bit True/Complement Zero/One Element 74LS87: Schema pratico

La corrente massima $\mathbf{I_{OL}}$ assorbita dalle uscite del **74LS87** è di **20 mA**, la potenza dissipata massima è di **445 mW** e il ritardo di propagazione massimo $\mathbf{t_{PLH}}$ e $\mathbf{t_{PHL}}$ (con carico di **280**ohm/**25**pF) è pari a **20 ns**.

La versione decimale (o meglio **BCD**) appartiene invece alla **serie CMOS**: si tratta dell'integrato **4561**, definito **BCD 9's Complementer** per evidenziare la sua attitudine a generare il *Complemento a 9* di un codice BCD proposto ai suoi ingressi; la *Figura 14* mostra il suo *pin-out*.

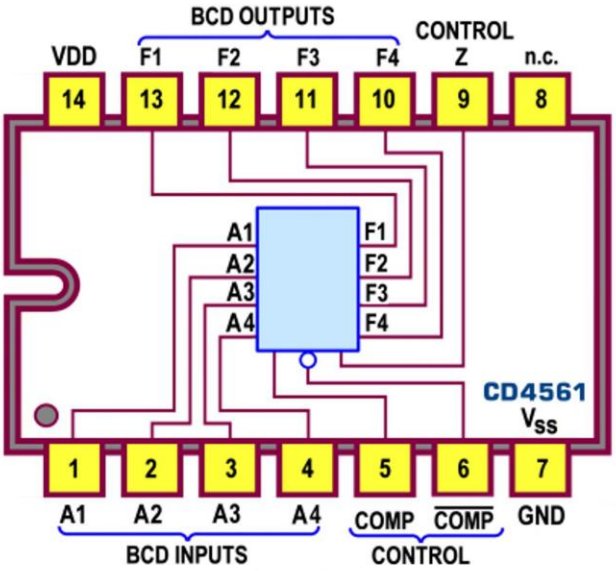


Figura 14 - BCD 9's Complementer 4561: Pin-out

Il componente dispone di 3 linee di controllo, in aggiunta alle 4 d'ingresso (sulle quali è prevista l'applicazione delle sole 10 parole di un codice a 4 bit di tipo BCD) e alle 4 uscite; lo *schema funzionale* originale (*Logic Diagram*) è piuttosto complicato, pur lasciando intuire le azioni necessarie per assicurare il servizio; per questa ragione ho preferito sostituirlo con uno molto intuitivo, ottenuto direttamente dalle formule di progetto relative alle 4 funzioni booleane descritte dalla Tabella di verità e dedotte dalle rispettive mappe di Karnaugh.

La *Figura 15* aiuta a capire il modo di operare del **4561**; la condizione principale per il funzionamento è che l'ingresso Z (*Zero Control*) sia forzato a **0** logico, tenendo aperte le porte **AND** e consentendo il passaggio in uscita dell'informazione creata a monte; in caso contrario (Z=**1**) tutte le uscite sono forzate a **0**.

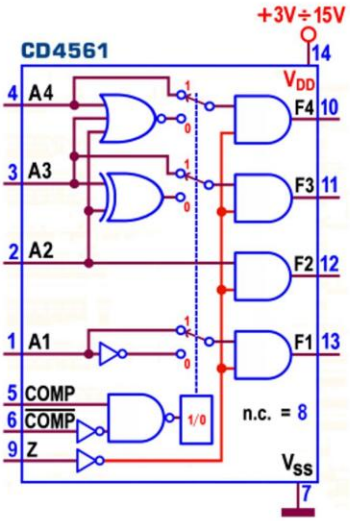
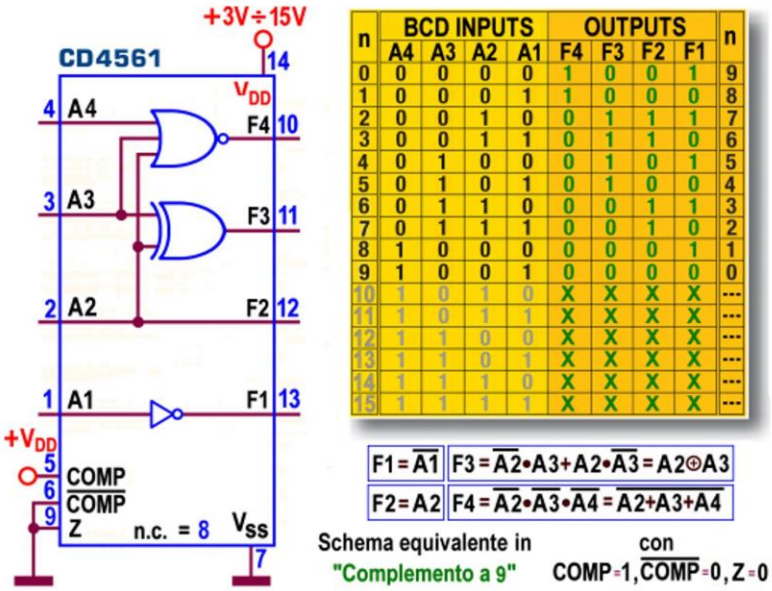


Figura 15 - BCD 9's Complementer 4561: Schema funzionale in sintesi

Spetta poi alle 2 linee *Complement Control* (pin 5 e pin6) stabilire il valore forzato sulle uscite: possiamo notare che esse intervengono sulla posizione **0** o **1** di un multiplexer interno e che

solo se COMP=**1** e $(COMP)_{negato}$=**0** sarà disponibile il *Complemento a 9* del codice BCD proposto in ingresso: in tutti gli altri casi quest'ultimo passerà (*Straight-through*) inalterato in uscita.



n	BCD INPUTS A4	A3	A2	A1	OUTPUTS F4	F3	F2	F1	n
0	0	0	0	0	1	0	0	1	9
1	0	0	0	1	1	0	0	0	8
2	0	0	1	0	0	1	1	1	7
3	0	0	1	1	0	1	1	0	6
4	0	1	0	0	0	1	0	1	5
5	0	1	0	1	0	1	0	0	4
6	0	1	1	0	0	0	1	1	3
7	0	1	1	1	0	0	1	0	2
8	1	0	0	0	0	0	0	1	1
9	1	0	0	1	0	0	0	0	0
10	1	0	1	0	X	X	X	X	---
11	1	0	1	1	X	X	X	X	---
12	1	1	0	0	X	X	X	X	---
13	1	1	0	1	X	X	X	X	---
14	1	1	1	0	X	X	X	X	---
15	1	1	1	1	X	X	X	X	---

Figura 16 - BCD 9's Complementer 4561: uso in Complemento a 9

La *Figura 16* si riferisce alle condizioni d'uso per disporre del *Complemento a 9*, evidenziando i soli componenti necessari per questa operazione, gli stessi per altro derivati dalle equazioni di progetto ottenute dalla Tabella di verità, offerta con lo schema per facilitare la verifica di tutti i possibili codici BCD.

Possiamo fare alcune considerazioni: a) la Tabella riporta (in grigio) anche le 6 combinazioni non appartenenti al codice BCD, evidenziando per esse altrettante condizioni d'indifferenza, in tutte le 4 uscite; questa presenza facilità non poco la minimizzazione delle funzioni booleane corrispondenti; b) il bit meno significativo del *Complemento a 9* (F_1) è uguale a quello del codice BCD (A_1) sottoposto ad inversione logica; c) il bit di peso 1 è uguale in entrambe le parole (F_2=A_2); d) i rimanenti bit, F_3 e F_4, sono ottenuti semplicemente con una **OREX** a 2 ingressi e una **NOR** a 3 ingressi.

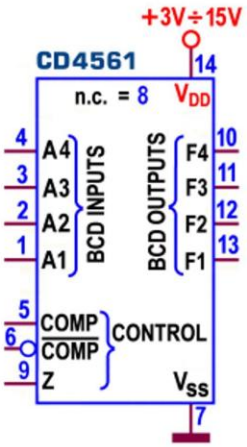


Figura 17 - BCD 9's Complementer 4561: Schema pratico

Lo *schema pratico*, consigliato nella stesura dei progetti per la sua sintetica completezza è proposto in *Figura 17*.

Come per gli altri componenti della famiglia **CMOS** la *potenza dissipata* è trascurabile mentre il *ritardo di propagazione massimo* (*Propagation Delay Time*, $\mathbf{t_{PHL}}$ e $\mathbf{t_{PLH}}$), rilevato con carico di **200**kohm/**50**pF, va da **1000ns** ($\mathbf{V_{DD}}$=**5V**) a **300ns** ($\mathbf{V_{DD}}$=**15V**).

Sommatore/Sottrattore

La sequenza di dispositivi **FS** in cascata non è la via realmente seguita per assicurare la *Sottrazione* di 2 numeri; oltre ad essere intrinsecamente lenta non esistono componenti integrati con questa struttura; la ragione sta nel fatto che essa può essere meglio espressa come somma del *Minuendo* con il *Complemento alla Base* del *Sottraendo*, scartando l'eventuale cifra più significativa del risultato, se dovesse avere una quantità di cifre maggiore rispetto a quella degli operandi (o dell'operando più grande).

Un esempio con numeri decimali ci aiuta a capire la validità della asserzione: 81-47 è (algebricamente) la stessa cosa di 81+(-47); per quanto abbiamo imparato il numero negativo -47 si può esprimere con il *Complemento a 10* di 47, per cui, in definitiva, avremo: $81+(47)_{Ca10}=81+53=134$, cioè appunto 34 (scartando la cifra 1 eccedente, detta *trabocco* o *overflow*).

Naturalmente questa tecnica vale anche in ambito binario; riprendiamo i dati trattati dal *Sottrattore Parallelo* di *Figura 6*, $(1100)_2$-$(1001)_2$; al *Minuendo* sommiamo allora il *Complemento a 2* del *Sottraendo*, cioè: $(1100)_2+(1001)_{Ca2}=(1100)_2+(0111)_2=(10011)_2$, cioè $(0011)_2=(3)_{10}$, scartando la cifra eccedente.

Tutto regolare: la tecnica funziona! Possiamo passare dalle parole ai fatti; basterà rispolverare uno dei sommatori analizzati in dettaglio le scorse puntate e applicare un circuito Complementatore al *Sottraendo*.

Possiamo cominciare con un *Ripple Carry Adder* fatto con 4 **FA** (*Full Adder*); ricordando che il *Complemento a 2* di un numero binario si ottiene aggiungendo **1** al suo *Complemento a 1* (cioè al numero ottenuto sostituendo gli **0** con degli **1**, e viceversa) per complementare a 1 il *Sottraendo* basteranno 4 inverter e per sommare **1** a questo risultato basterà utilizzare l'ingresso di *Riporto* R_0 del **FA** meno significativo: geniale!

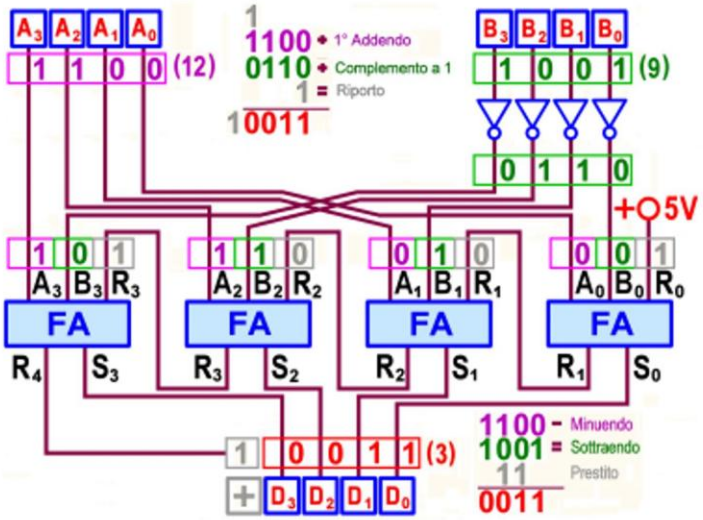


Figura 18 - Sottrattore con un 4-bit Ripple Carry Adder

In accordo con le premesse, ai fini del valore della *Differenza*, l'uscita di *Riporto* R_4 del **FA** più significativo verrà scartata (o utilizzata per stabilire il segno del risultato).

La *Figura 18* mostra il progetto appena descritto; il sommatore *Ripple Carry* può essere sostituito con un **7482** o con un più veloce **74LS83** o **74LS283**; in ogni caso, data la presenza fisica di un sommatore, è un peccato non rendere disponibili entrambe le operazioni di somma e di differenza.

Niente di più facile! Sostituendo i 4 inverter con altrettante porte **OREX** sarà infatti possibile controllare il secondo addendo del sommatore con una linea dedicata, chiamata a gestire anche la presenza o meno del *Riporto* R_0 del **FA** meno significativo.

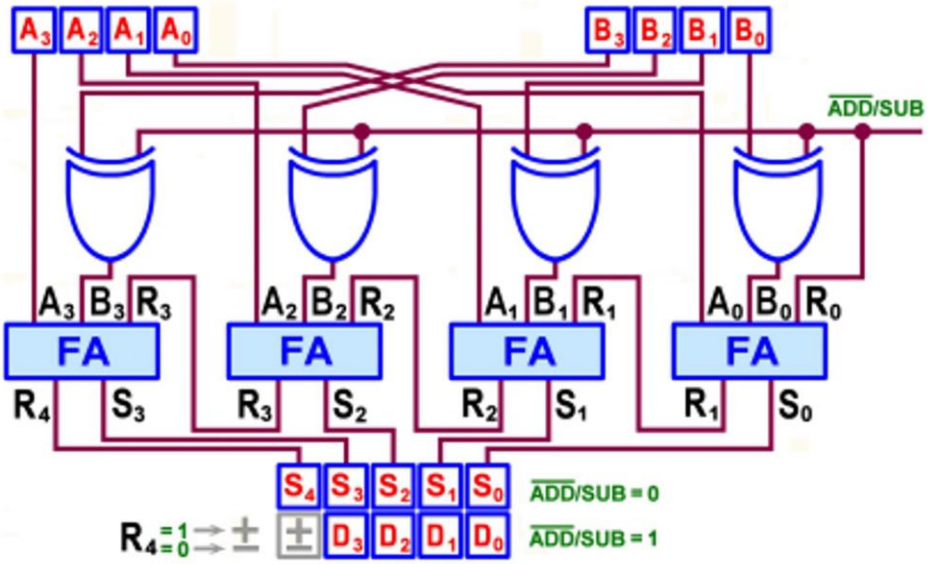


Figura 19 - 4-bit Adder/Subtractor: schema funzionale

La *Figura 19* mette in evidenza che, se la linea di controllo ADD/SUB è forzata a **0** il circuito si comporta da *Sommatore* A+B (le 4 **OREX** lasciano passare inalterati i bit di B e forzano a **0** il *Riporto* R_0) mentre se è posta a **1** si comporta da *Sottrattore* A-B (le 4 **OREX** invertono i bit di B e forzano a **1** il *Riporto* R_0, entrambe condizioni necessarie per trasformare il secondo operando nel suo *Complemento a 2*).

Come si vede il *Riporto* R_4 fornisce il bit5 del risultato se il dispositivo funziona da *Sommatore*, rimanendo irrilevante se funziona da *Sottrattore*, fermo restando che in questo caso potrà essere utilizzato per stabilirne il segno e (se negativo) per attivare la sua eventuale correzione.

In fatti, anche con questa soluzione va sottolineato che il risultato sarà corretto solo se il *Minuendo* A è maggiore o uguale al *Sottraendo* B (A>=B), altrimenti (A<B) sarà necessario intervenire, forzando su di esso l'operazione di *Complemento a 2*, invertendone il valore e sommando **1**.

Per fissare le idee: con *Minuendo* $A=(1001)_2=(9)_{10}$ e *Sottraendo* $B=(1100)_2=(12)_{10}$ il dispositivo di *Figura 19* darà come risultato $(1001)_2+(1100)_{Ca2}=(1001)_2+(0100)_2=(1101)_2$ con R_4=**0** (sinonimo di segno negativo) per cui dovrà essere attivato un ulteriore hardware a valle (non visibile in figura) e il risultato effettivo sarà $(1101)_{Ca2}=(0011)_2=(3)_{10}$ che, in virtù del segno R_4=0, è proprio (-3).

In concreto (vedi *Figura 20*) sarà sufficiente un *4-bit Fast Carry Full Adder* (come il **74LS283** o il **74LS83**) e un **74LS86** (4 **OREX** a 2 ingressi); poiché non è opportuno lasciare fluttuanti

(scollegati) gli ingressi la linea di controllo ADD/SUB sarà fissata al positivo dell'alimentazione con un resistore da 10kOhm, predisponendo in questo modo il circuito a funzionare come *Sottrattore* e richiedendo esplicitamente uno **0** per operare la *Somma* degli operandi.

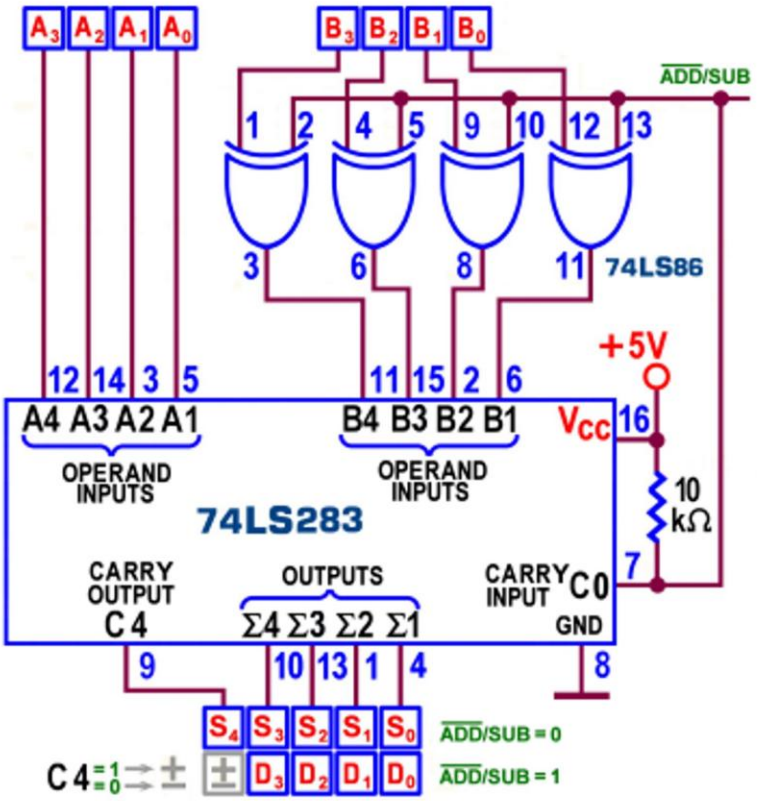


Figura 20 - 4-bit Adder/Subtractor: progetto con 74LS283

Naturalmente per il *Carry Output* C_4 valgono le considerazioni fatte per il *Riporto* R_4 della figura precedente, compresa la necessità di ulteriore hardware per forzare il *Complemento a 2* del risultato, se il progetto è chiamato a gestire la *Differenza* di un *Minuendo* A minore del *Sottraendo* B.

Il circuito di *Figura 20* può essere duplicato **n** volte per realizzare sommatori/sottrattori a **n***4 bit, collegando l'uscita C_4 di ciascun sommatore con l'ingresso C_0 di quello successivo e controllando ciascuna batteria di **OREX** con la stessa linea ADD/SUB.

La possibilità di realizzare i *Sottrattori* con i *Sommatori* ha reso molto povera la disponibilità di integrati specifici; tra essi è nota la presenza del **10180**, un **Dual 2-Bit Adder/Subtractor** in tecnologia **ECL**, capace di garantire 4 diverse operazioni sui suoi 2 operandi A e B, ciascuno a 1 bit, in funzione del valore delle 2 linee di controllo S_0 e S_1 (in comune ad entrambi) lasciandone disponibile il risultato sia in forma diretta che negata.

Ciascuno dei 2 dispositivi può disporre delle proprie linee di Riporto, sia d'ingresso che d'uscita; la *Figura 21* mostra il suo *pin-out*.

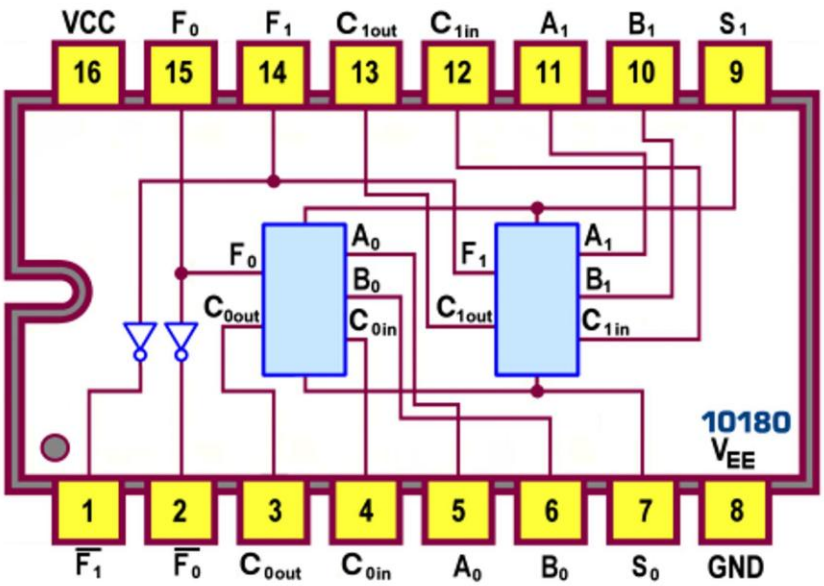
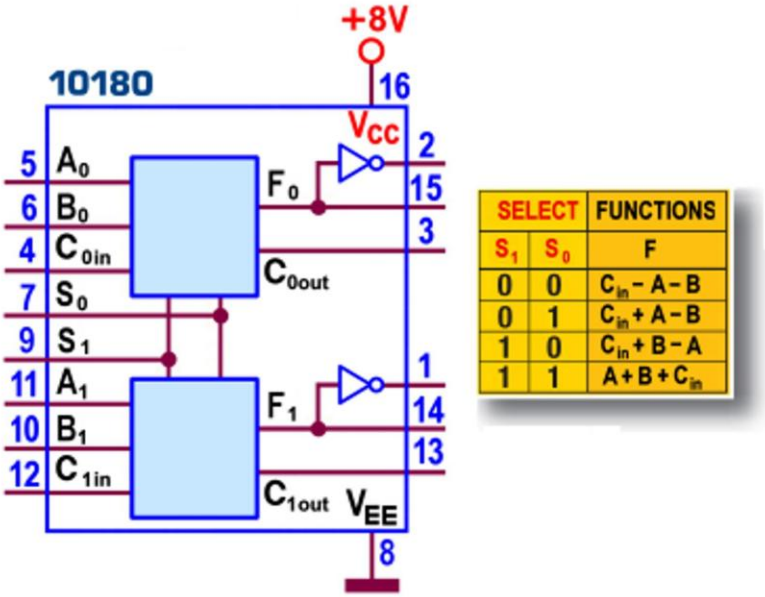


Figura 21 - Dual 2-Bit Adder/Subtractor 10180: Pin-out



SELECT		FUNCTIONS
S_1	S_0	F
0	0	$C_{in} - A - B$
0	1	$C_{in} + A - B$
1	0	$C_{in} + B - A$
1	1	$A + B + C_{in}$

Figura 22 - Dual 2-Bit Adder/Subtractor 10180: Schema funzionale

Progettato per creare moltiplicatori ad alta velocità dispone di un resistore di pull-up integrato per ciascuna delle linee d'ingresso, per cui quelle non utilizzate possono essere lasciate aperte senza problemi; lo *schema funzionale* è disponibile in *Figura 22*.

Il **10180** è caratterizzato da in ritardo di propagazione massimo di 5,4 ns e da un assorbimento tipico di 70 mA.

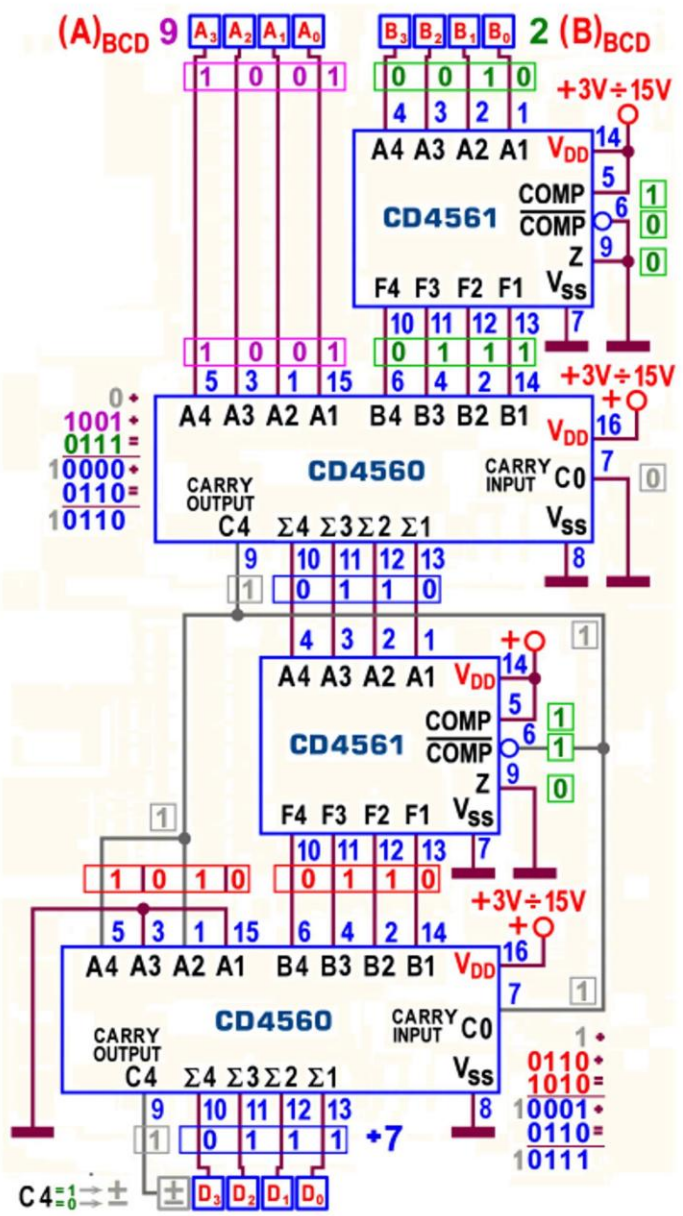


Figura 23 - Sottrattore BCD con 4560/4561

Pur essendo logicamente differenti (sono macchine sequenziali, non combinatorie, come quelle trattate finora) possiamo citare anche il componente **TTL 74385**, definito **Quadruple Serial Adders/Subtractors** dai datasheet; contiene 4 dispositivi indipendenti, chiamati a Sommare/Sottrarre i bit di dato proposti su altrettanti ingressi, in corrispondenza del fronte di salita di un segnale di clock, in comune a tutti come la linea di Clear, necessaria per inizializzare a 0, all'occorrenza, i processi aritmetici.

Questi integrati sono spesso associati al **74384** (**Quadruple Serial/Parallel Two's Complement Multiplier**).

Sommatore/Sottrattore BCD

In chiusura di questa rassegna non può mancare un angolino dedicato alla *Sottrazione* di numeri espressi in codice BCD; la scorsa puntata abbiamo conosciuto il *Sommatore* **NBCD Adder 4560**, dotato di circuiti di tipo ***Carry Look Ahead*** per la determinazione veloce del Riporto; in questa ci siamo occupati del **BCD 9's Complementer 4561**, indispensabile per consentire al primo di operare la sottrazione.

La *Figura 23* mette in evidenza tutti i passaggi imposti sui numeri in ingresso, per esempio per sottrarre B=$(0010)_{BCD}$=$(2)_{10}$ da A=$(1001)_{BCD}$=$(9)_{10}$; il primo **4561** è programmato (COMP=**1**, $COMP_{negato}$=**0**) per trasformare B nel suo *Complemento a 9,* $(0010)_{Ca9}$=$(0111)_2$, a beneficio della successiva somma con A; il risultato, $(0110)_2$=$(6)_{10}$, è lasciato inalterato dal secondo **4561** (*Straight-through*, essendo a **1** entrambi i *Complement Control*) a beneficio del secondo sommatore chiamato a fissare il risultato (aggiunto al *Riporto* precedente) e il segno della *Differenza*.