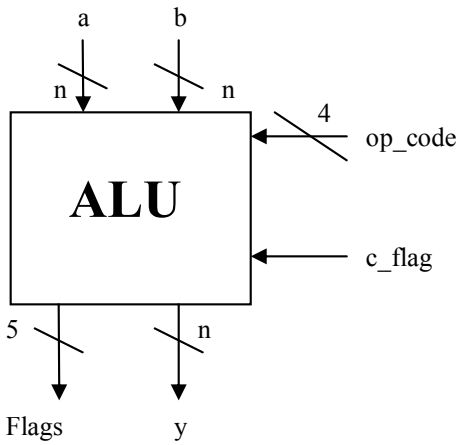
GIOVANNI DI CECCA & VIRGINIA BELLINO
50 / 887 408 / 466

PROGETTAZIONE E SIMULAZIONE DI UNA ALU COMPLESSA CON PROGRAMMI



http://www.dicecca.net

Indice

Introduzione 4

Progetto ALU 7
Cenni preliminari 8
Interfaccia ALU 9
Decoder 12
Logic_Unit 14
Arith_Unit 16
Modify_A 18
Modify_B 20
Bin_adder 22
Full ed Half adder 24
Flag_unit 28
Carry_flag 30
Sign_flag 32
Parity_flag 34
Overflow_flag 36
Zero_flag 38

Programmi per l'ALU 40
Cenni preliminari 41
Forma base $c^2=a^2+b^2$ 42
Codice Matlab valori incorporati 43
Codice Matlab valori da tastiera 44
Forma completa 47
Codice Matlab valori da tastiera 48

Multiplexer 50
Cenni preliminari 51
Mux 16 a 1 52
Mux 8 a 1 54
Mux 4 a 1 56
Mux 2 a 1 58

Esempi d'uso 60
Test Mux 16 a 1 61
Test Mux 8 a 1 62
Test Mux 4 a 1 63
Test Mux 2 a 1 64

Appendici 65

Introduzione

In questo volume sono racchiusi i progetti realizzati durante il corso di Laboratorio di Architettura degli elaboratori nell'anno accademico 2002 / 2003.

Di particolare interesse è il progetto di un'ALU che permette di realizzare semplici calcoli.

Per la realizzazione delle molteplici componenti è stato seguito un approccio di tipo **TOP DOWN**.

Tutte le strutture logiche sono ampiamente documentate e accompagnate da programmi simulativi realizzati in MATLAB®.

Per la simulazione effettiva abbiamo usato **MATLAB Student Version 6.0**

Gli autori

Progetto

di

un'ALU

Complessa

Cenni preliminari

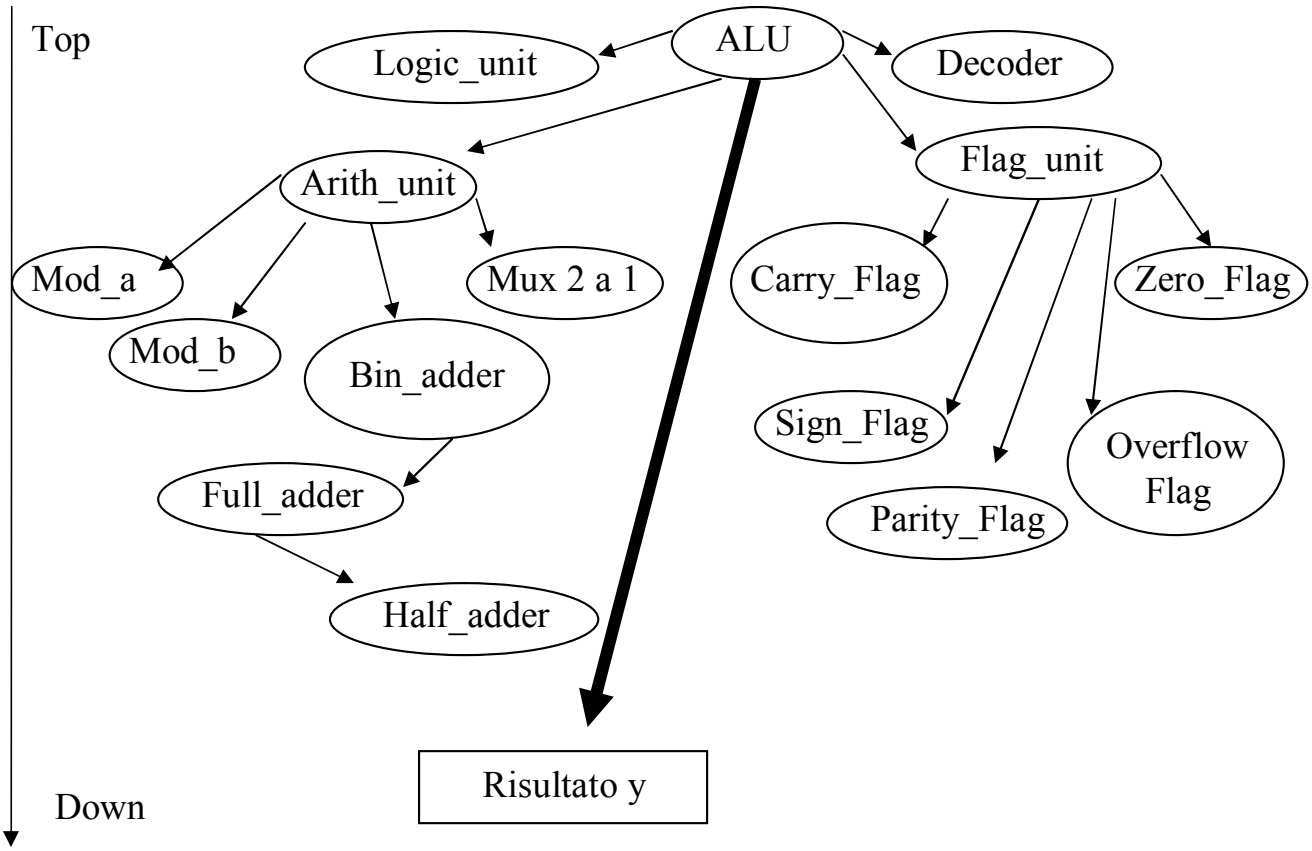
Il presente progetto analizza il funzionamento a livello hardware di un'ALU Complessa.

Esso è stato diviso in diversi moduli che esaminano il funzionamento delle singole parti del progetto, e ciascun modulo è stato opportunamente testato.

I moduli presenti sono:

- Unità aritmetica
- Unità logica
- Flag unit
- Decoder
- Multiplexer

Schema del progetto:



ALU

L'interfaccia grafica ALU corrispondente al livello 1 mostra immediatamente all'utente quali sono i dati che l'alu riceve dall'esterno, e quali risultati essa produce in uscita.

Più precisamente avremo:

a, b: operandi

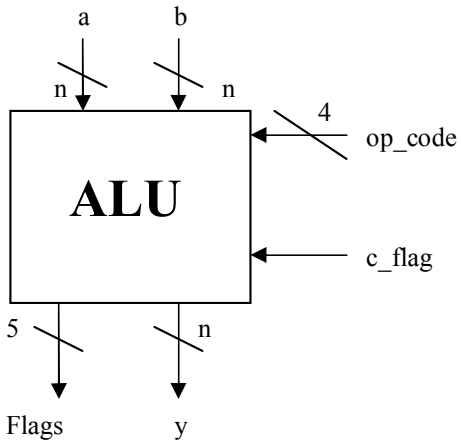
op_code: codice che identifica l'istruzione da eseguire

c_flag: riporto di ingresso

y: risultato

flags: segnalatori

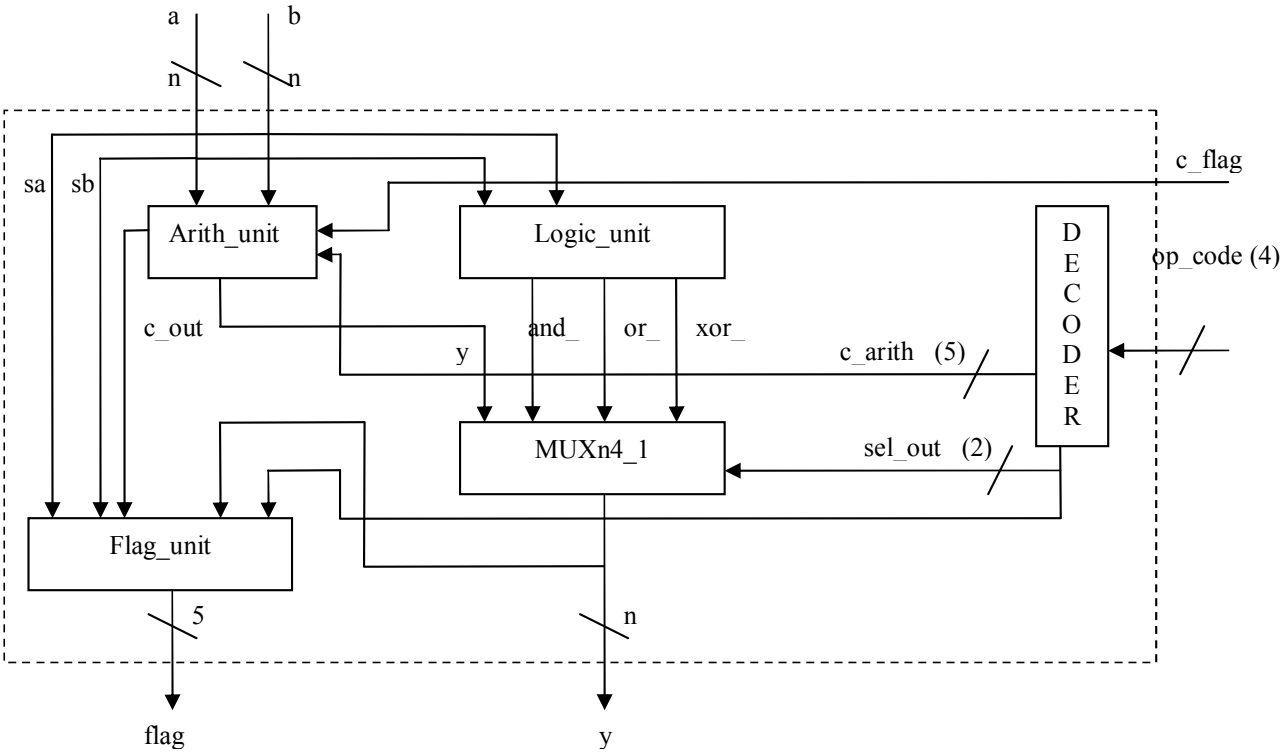
Livello 1



Il livello 2 fornisce invece maggiori dettagli sulla struttura interna dell'ALU, mostrando i diversi sottoinsiemi di cui essa si compone.

Tali sottoinsiemi sono poi ampiamente illustrati e documentati in seguito.

Livello 2



L'ALU appena descritta sarà in grado di eseguire le seguenti istruzioni:

- **ADD**: addizione tra interi
- **ADC**: addizione con riporto
- **SUB**: sottrazione
- **SUBB**: sottrazione con "prestito"
- **INCb**: incremento
- **DECa**: decremento
- **NEGb**: negazione
- **NOTb**: not
- **AND**: and
- **OR**: or
- **XOR**: xor

Le prime 6 istruzioni, insieme a Not e Neg vengono eseguite dall'unità aritmetica, mentre AND, OR, e XOR vengono eseguite dall'unità logica.
La funzione XOR inoltre, funge anche da comparatore.

Programma simulativo

Il seguente programma, realizzato in Matlab, simula il funzionamento dell'ALU.

```
%            Progetto ALU
%
%       Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%       http://www.dicecca.net

% Interfaccia utente della ALU
function [y,flags]=alu(a,b,op_code,c_flag)

% Memorizza il segno degli operandi a et b
sa=a(1);
sb=b(1);

% Carica il decoder e passa i dati dell' OP_CODE
[c_arith,ncmp,sel_out]=decoder(op_code);

% Carica l'unità logica aritmetica
[and_,or_,xor_]=logic_unit(a,b);

% Carica l'unità aritmetica
[y,c_out]=arith_unit(a,b,c_arith,c_flag);

% Memorizza il risultato del MUX 4_1
y=muxn4_1(y,and_,or_,xor_,sel_out);

% Associa alla variabile m_out il valore del muxn4_1 per poi
% passarlo alle flags
m_out=y

% Carica la Flag Unit
[flags]=flag_unit(c_out,sa,sb,m_out,ncmp,sel_out);
```

Esempio d'uso

Nell'**Appendice** vi sono i tabulati stampati direttamente da Matlab

Decoder

Il decoder permette di decodificare il codice operativo identificando cosi il tipo di istruzione da eseguire.

I relativi codici sono riportati nella seguente tabella di verità

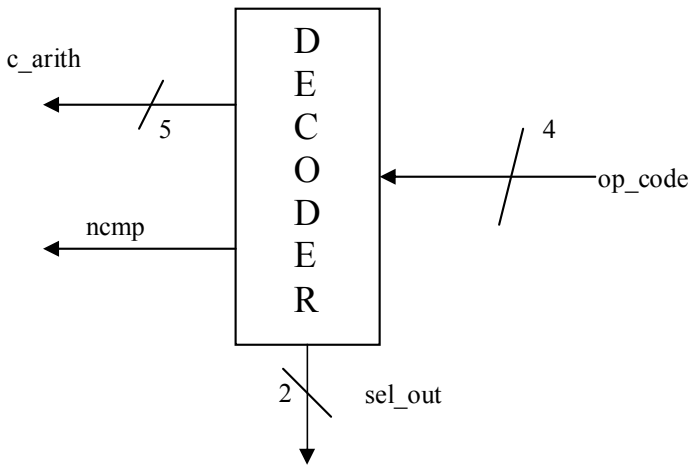
TABELLA DI VERITÀ

Istruzione	op_code	c_or	c_xor	c_and	sel_c	c_in	ncmp	sel_out
ADD	0000	0	0	1	0	0	1	00
ADC	0001	0	0	1	1	-	1	00
SUB	0010	0	1	1	0	1	1	00
SUBB	0011	0	1	1	1	-	1	00
INCb	0100	0	0	0	0	1	1	00
DECa	0101	1	0	1	0	0	1	00
NEGb	0110	0	1	0	0	1	1	00
NOTb	0111	0	1	0	0	0	1	00
AND	1000	-	-	-	-	-	-	01
OR	1001	-	-	-	-	-	-	10
XOR	1010	-	-	-	-	-	-	11

Le variabili utilizzate sono:

- op_code: codice operativo immesso per eseguire la computazione
- c_arith: codici che il decoder passa all'unità aritmetica
- n_cmp: variabile che viene passata alla flag unit
- sel_out: bit di controllo del mux 4 a 1 di uscita dati

Graficamente avremo:



Codice di simulazione Matlab

```
%            Progetto ALU
%
%       Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%       http://www.dicecca.net

% decodificatore del codice operativo
function [c_arith,ncmp,sel_out]=decoder(op_code)

% Definisci valori dei quattro ingressi e f c d
e=op_code(1);
f=op_code(2);
c=op_code(3);
d=op_code(4);

% assenga valori alle variabili
c_and=(~e&d)|(~f&~c)|(~f&~d);
c_or =~e&f&~c&d;
c_xor=~e&c;
sel_c=~e&~f&d;
c_in =(~e&c&~d)|(~e&f&~d)|(~e&c&d);

% Fornisci i risultati di uscita
c_arith=[c_and,c_or,c_xor,sel_c,c_in];
ncmp=~e;
sel_out(1)=(e&~f)&(xor(c,d));
sel_out(2)=e&~f&~d;
```

Logic_unit

L'unità logica permette di eseguire le operazioni logiche AND, OR, XOR ricevendo in ingresso i due operandi a e b.

Graficamente avremo:

Livello 1

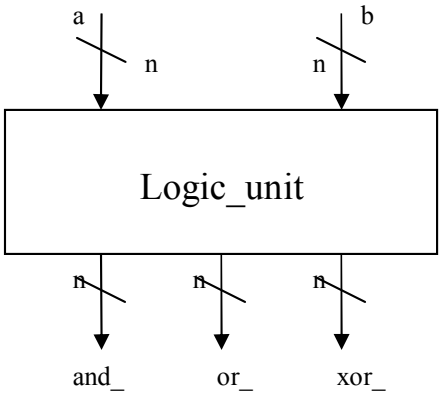
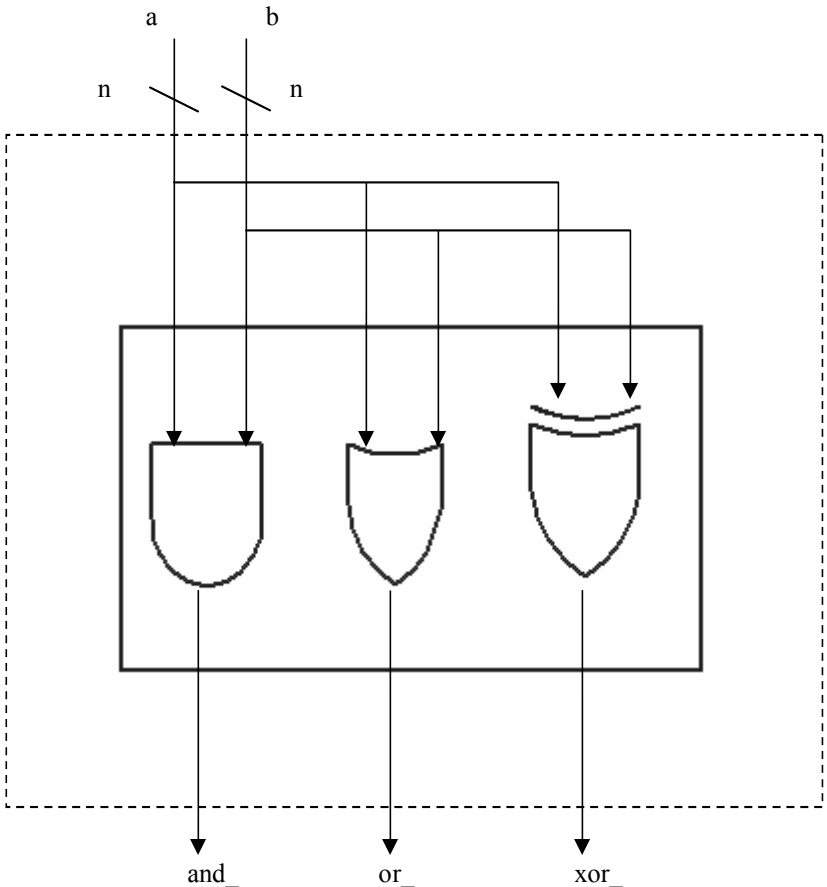


Tabella di verità

a (i)	b(i)	and_	or_	xor_
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Livello 2



Codice di simulazione Matlab

```
%            Progetto ALU
%
%       Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%       http://www.dicecca.net

% Unità logica, effettua l'AND, l'OR e lo XOR dei rispettivi bit di a e b
function [and_, or_, xor_]=logic_unit(a,b)

% Calcola la lunghezza di a
n=length(a);

% Calcola i valori and or xor
for i=1:n
   and_(i)=a(i)&b(i);
   or_(i)=a(i)|b(i);
   xor_(i)=xor(a(i),b(i));
end
```

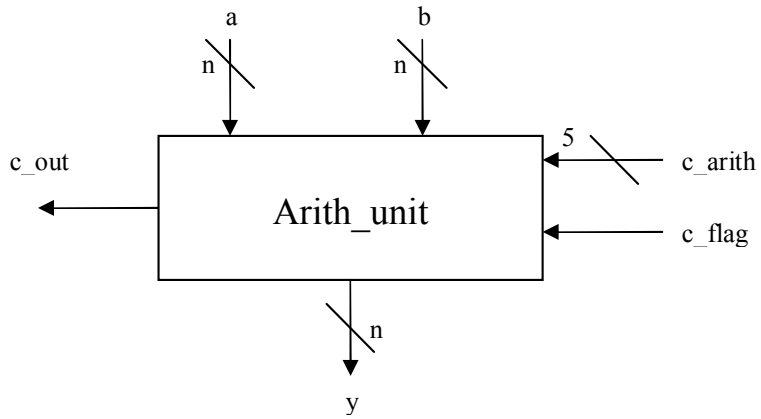
Arith_unit

L'unità Aritmetica permette di effettuare calcoli aritmetici sugli operandi a e b utilizzando le seguenti variabili:

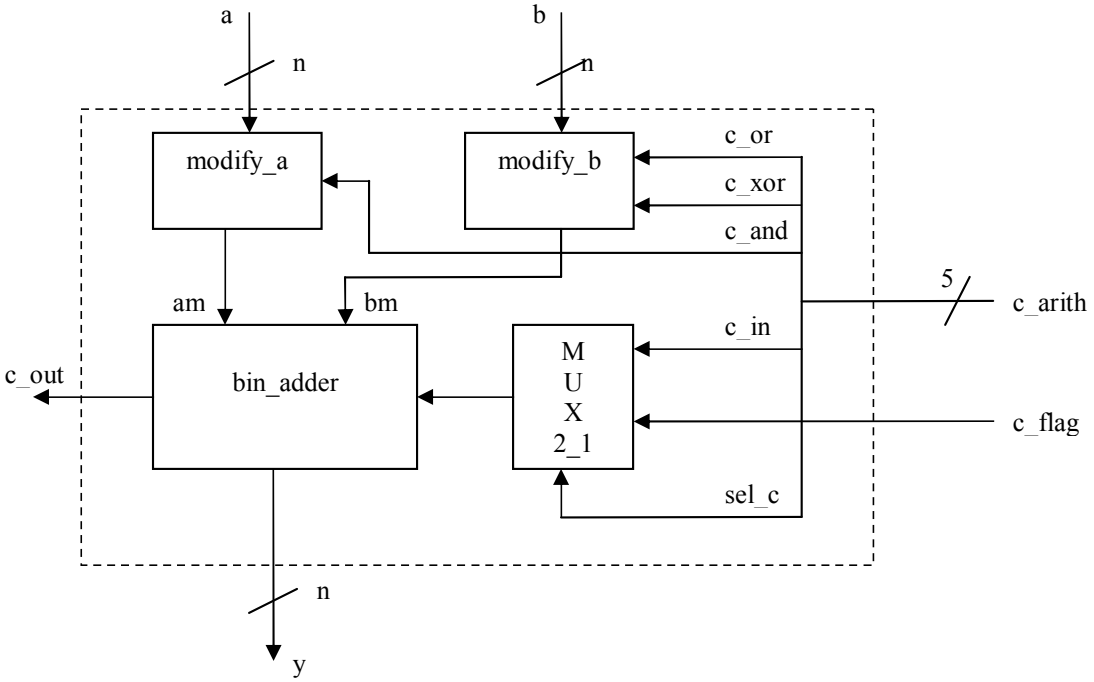
- C_arith: sono i 5 bit che il decoder attiva in uscita
  `c_arith=[ c_and, c_or, c_xor, sel_c, c_in ]`
- C_flag: Carry flag in ingresso
- C_out: Carry out

Graficamente avremo:

Livello 1



Livello 2



Codice di simulazione Matlab

```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net

% Unità aritmetica (è in grado di eseguire addizioni e sottrazioni)
function [y,c_out]=arith_unit(a,b,c_arith,c_flag)

% Associa alle variabili il valore di C_ARITH a 5 bit fornito dal decoder
c_and=c_arith(1);
c_or =c_arith(2);
c_xor=c_arith(3);
sel_c=c_arith(4);
c_in =c_arith(5);

% Calcola i valori di a et b modificati
am=modify_a(a,c_and);
bm=modify_b(b,c_or,c_xor);

% Calcola il nuovo Carry In
c_in=muxn2_1(c_in,c_flag,sel_c);

% Calcola il valore di a et b mediante un Addizionatore binario
[y,c_out]=bin_adder(am,bm,c_in);
```

Modify_A

Il modificatore serve, eventualmente, a modificare il valore dell'operando a,

Livello 1

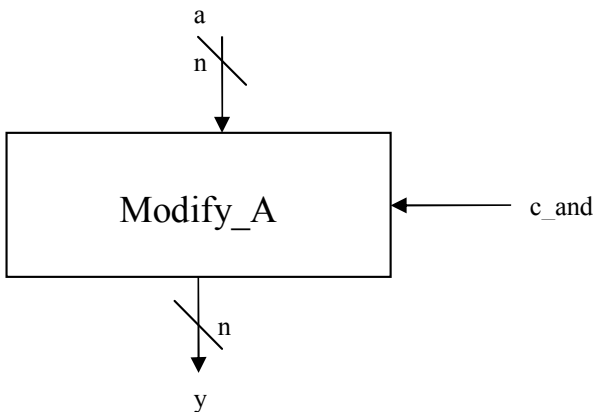
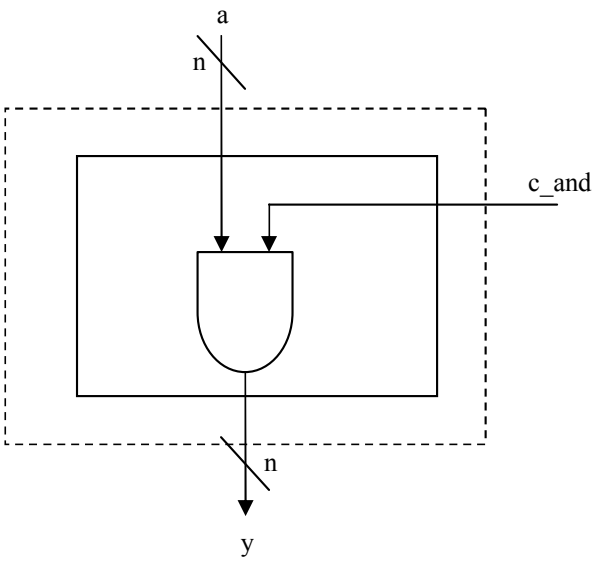


TABELLA DI VERITÀ

c_and	y
0	0
1	a

Livello 2



Codice di simulazione Matlab

```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net


% modulo di modifica di a
function y=modify_a(a,c_and)

% Calcola la lunghezza di a
n=length(a);

% Modifica i bit di a
for i=1:n
   y(i)=a(i)&c_and;
end
```

Modify_B

Il modificatore serve, eventualmente, a modificare il valore dell'operando b.

Livello 1

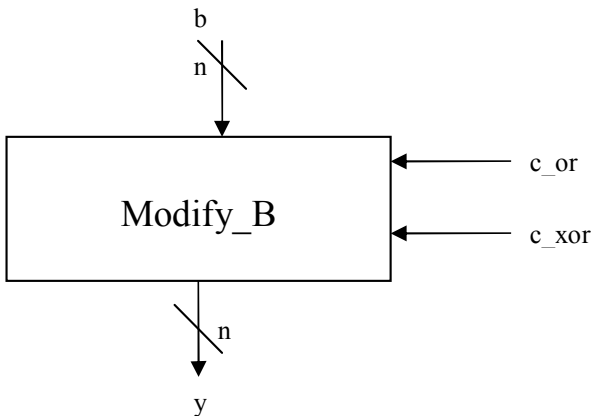
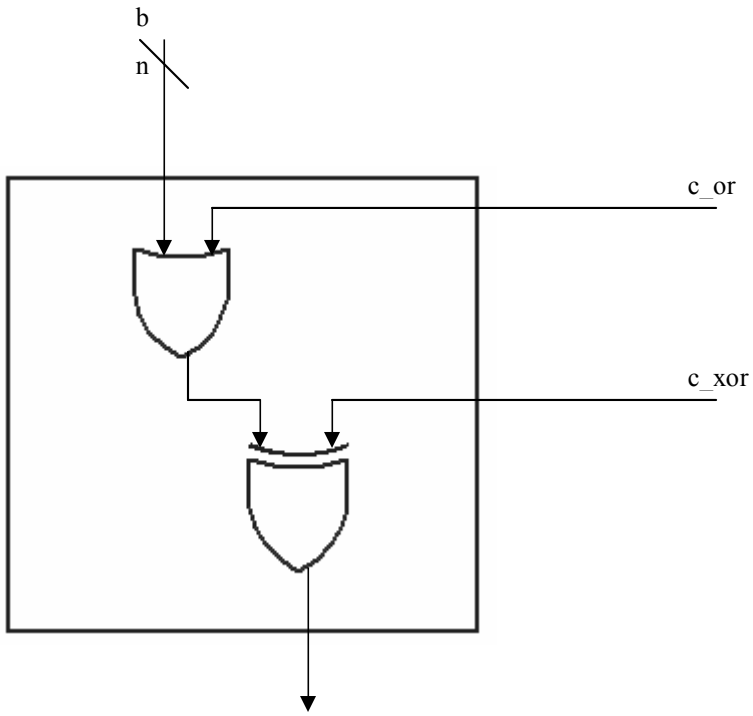


TABELLA DI VERITÀ

c_or	c_xor	Y
0	0	b
0	1	~b
1	0	-1
1	1	0

Livello 2



Codice di simulazione Matlab

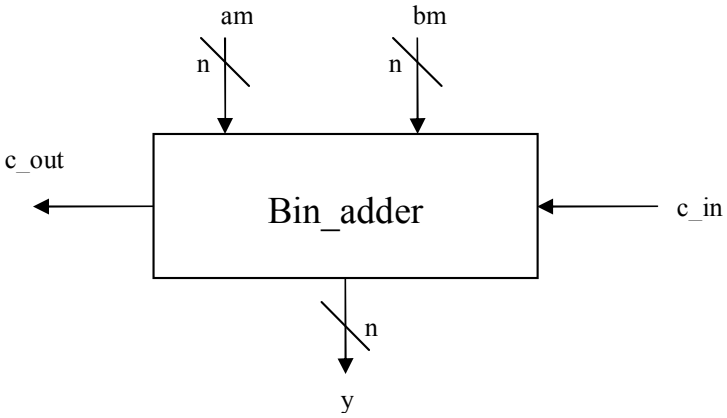
```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net


% modulo di modifica di b
function y=modify_b(b,c_or,c_xor)

% Calcola la lunghezza di b
n=length(b);

% Modifica i bit di b
for i=1:n
   s(i)=b(i)|c_or;
   y(i)=xor(s(i),c_xor);
end
```

Bin_adder

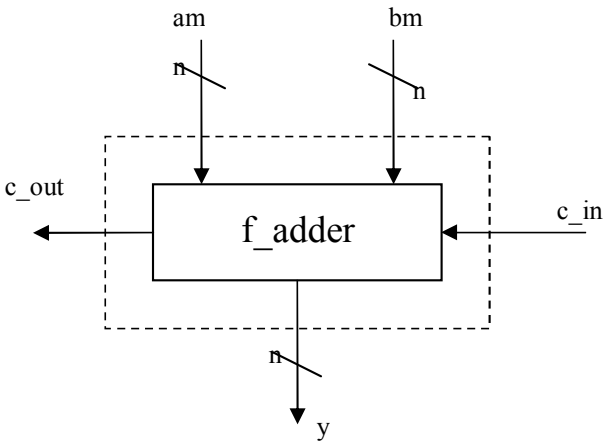
Il Bin adder esegue i calcoli sugli operandi immessi. Esso si compone di n-1 sommatori completi (full_adder) e di un semi-sommatore (half-adder).

Graficamente avremo:

Livello 1



Livello 2



Codice di simulazione Matlab

```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net


% Addizionatore binario
function [y,c_out]=bin_adder(a,b,c_in)

% Calcola la lunghezza del vettore a
n=length(a);

% Associa al Carry in uscita il valore di quello in ingresso
c_out=c_in;

% Calcola i valori di a et b con un sommatore completo bit a bit
for i=n:-1:1
   [y(i),c_out]=f_adder(a(i),b(i),c_out);
end
```

F_adder e half - adder

In seguito viene dettagliatamente illustrato il funzionamento delle due unità che compongono il bin adder.

La sostanziale differenza esistente tra un full ed un half adder sta nef fatto che il secondo non prevede un riporto di ingresso.

Per il full adder, graficamente avremo:

Livello 1

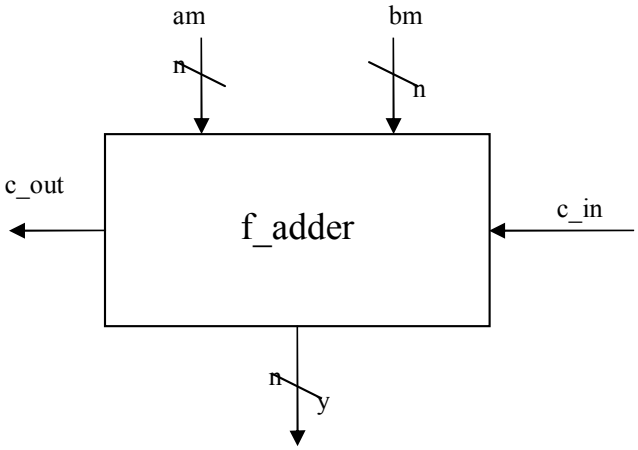
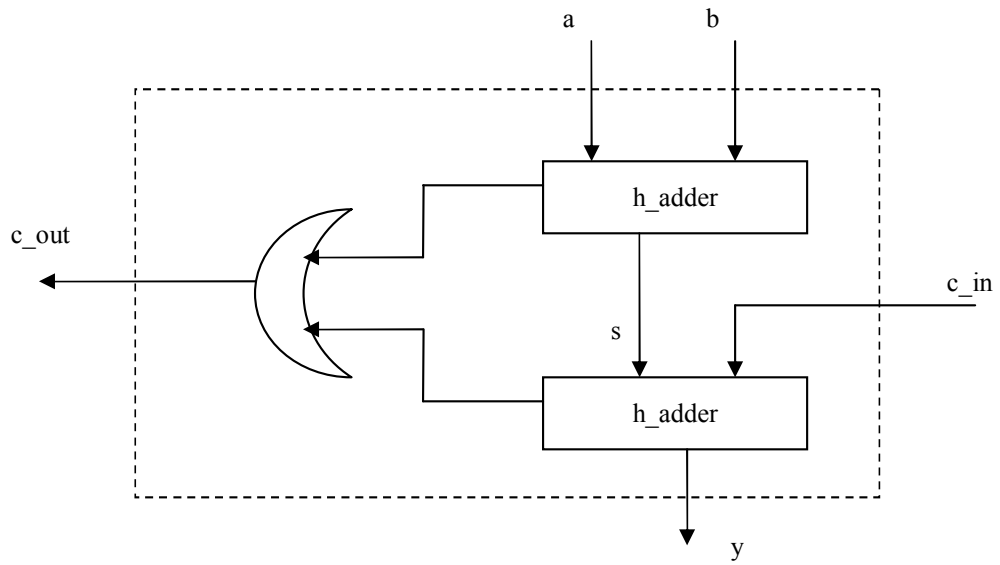


TABELLA DI VERITÀ

c_in	a	b	y	c_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Livello 2

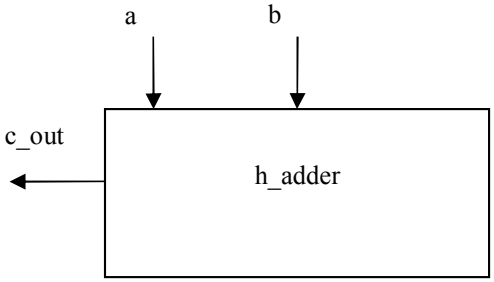


Codice di simulazione Matlab

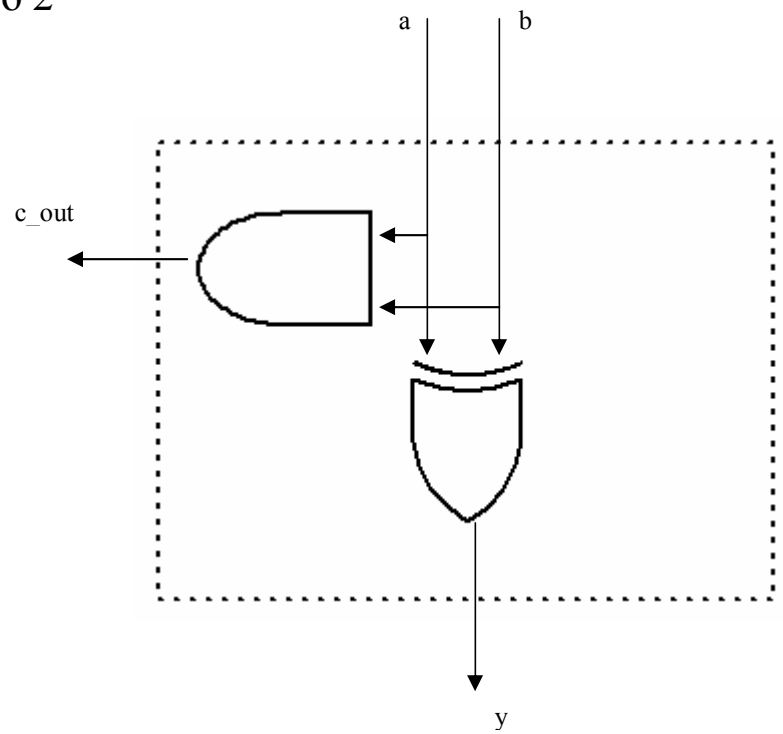
```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net


% addizionatore completo bit a bit (accetta riporto in ingresso)
function [y,c_out]=f_adder(a,b,c_in)

% Passa i valori ad un semisommatore
[s,c1]=h_adder(a,b);
[y,c2]=h_adder(s,c_in);

% Valuta se il c_out proviene da c1 o c2
c_out=c1|c2;
```

Per l' half adder invece, graficamente avremo:

Livello 1



Livello 2



Codice di simulazione Matlab

```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%         http://www.dicecca.net


% Semiaddizionatore bit a bit
function [y,c_out]=h_adder(a,b)

% Usando il comparatore
y=xor(a,b);

% Calcola il valore del CARRY OUT
c_out=a&b;
```

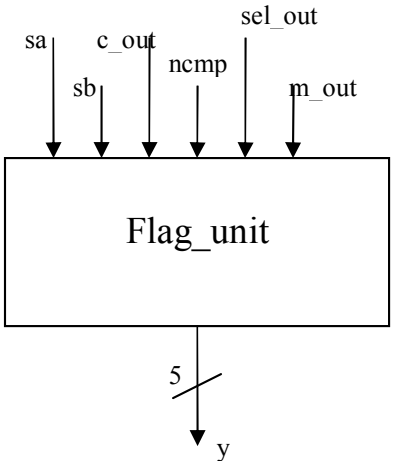
Flag_unit

La flag_unit è la parte di controllo della intera ALU, ed effettua ad ogni passo dell'elaborazione i controlli sulle tipologie di risultati ottenuti.

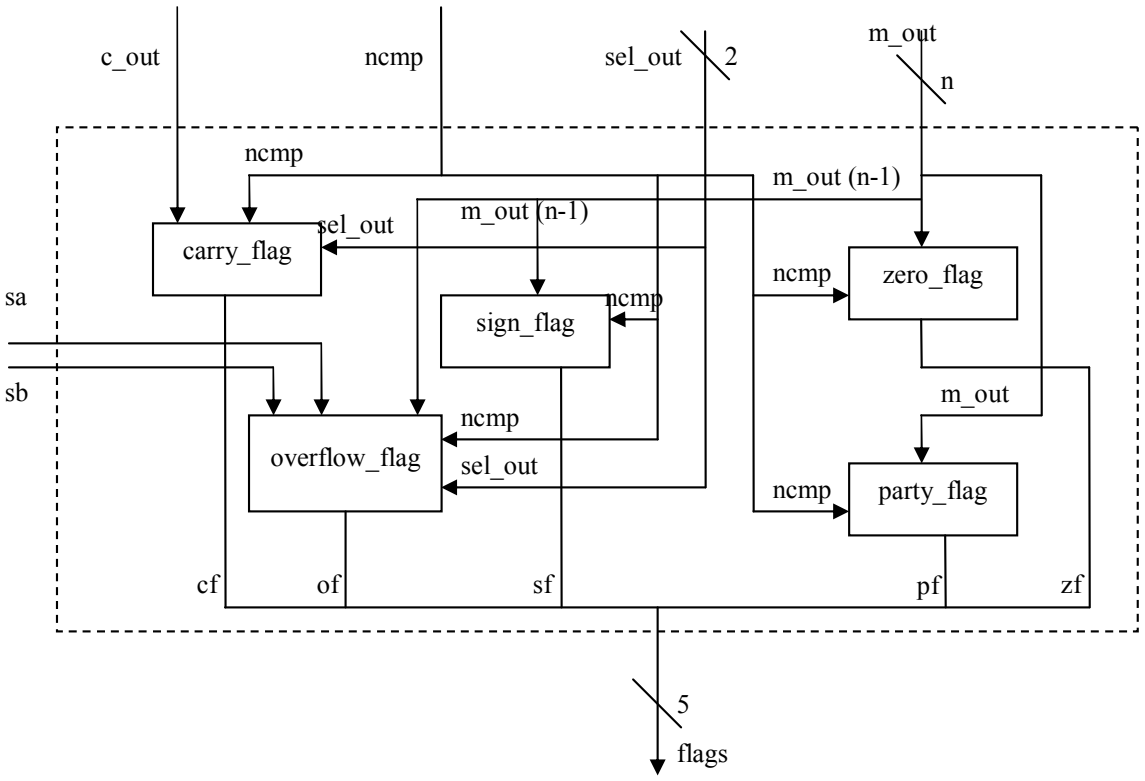
Essa è composta da: carry_flag, sign_flag, overflow_flag, zero_flag, parità_flag.

Da in uscita 5 bit che poi vengono visualizzati direttamente durante l'output

Livello 1



Livello 2



Codice di simulazione Matlab

```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net


% unità che raccoglie le flag dell'ALU
function [flags]=flag_unit(c_out,sa,sb,m_out,ncmp,sel_out)

% Associa i 5 bit della flag i risultati passati dalle funzioni sulle flag
flags(1)=carry_flag(c_out,ncmp,sel_out);
flags(2)=sign_flag(ncmp,m_out(1));
flags(3)=parity_flag (ncmp,m_out);
flags(4)=overflow_flag (ncmp,m_out(1),sa,sb,sel_out);
flags(5)=zero_flag(ncmp,m_out);
```

Carry_flag

Il carry_flag controlla il riporto di un'operazione. È posta ad 1 quando un operazione aritmetica genera un riporto o un prestito sul bit più significativo del risultato, altrimenti è 0. Questa flag indica una condizione di overflow per operazioni aritmetiche su operandi interi senza segno

Livello 1

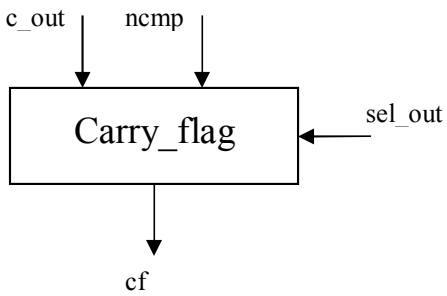
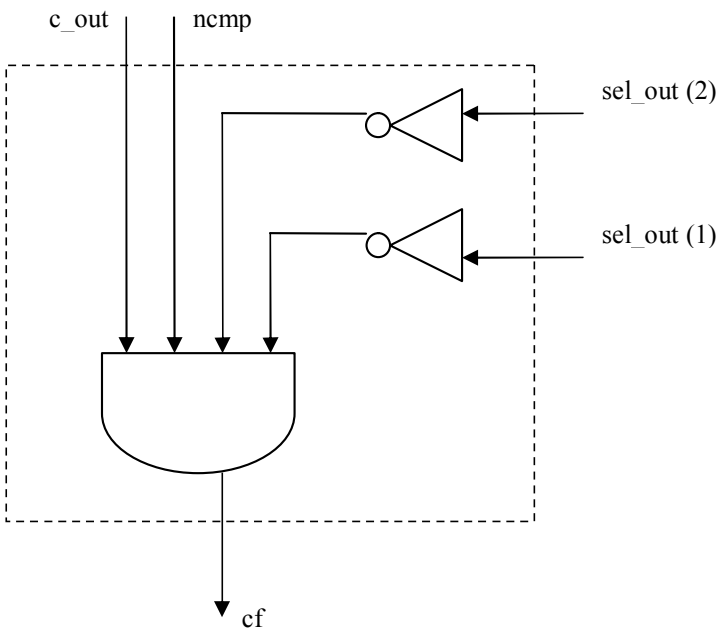


TABELLA DI VERITÀ

ncmp	sel_out	cf
1	00	c_out
0	00	0
X	01	0
X	10	0
X	11	0

Livello 2



Codice di simulazione Matlab

```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net


% cf assume il valore della linea c_out del bin-adder contenuto nell'unità
% aritmetica se è stata richiesta un'operazione aritmetica ad esclusione di CMP
function cf=carry_flag(c_out,ncmp,sel_out)
cf=(c_out)&(ncmp)&(~sel_out(1))&(~sel_out(2));
```

Sign_flag

Il valore di questo flag è uguale al valore del bit più significativo del risultato dell'ultima operazione aritmetica. Esso corrisponde al bit del segno in un numero intero dotato di segno. (0 indica un valore positivo, 1 un valore negativo).

Livello 1

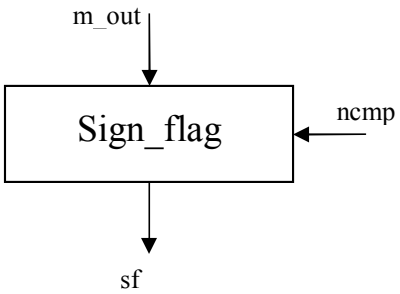
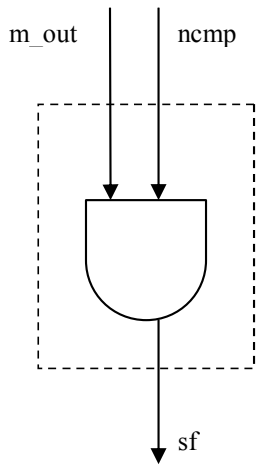


TABELLA DI VERITÀ

ncmp	sf
1	m_out
0	0

Livello 2



Codice di simulazione Matlab

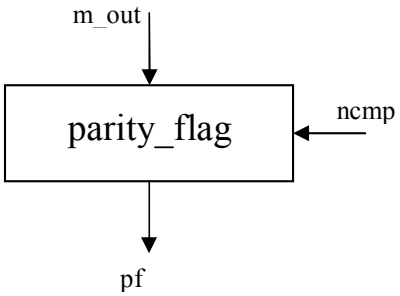
```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net


% sf assume il valore del segno del risultato quando non è stata effettuata
% un'operazione di CMP
function sf=sign_flag(ncmp,m_out)

sf=ncmp&m_out;
```

Parity_flag

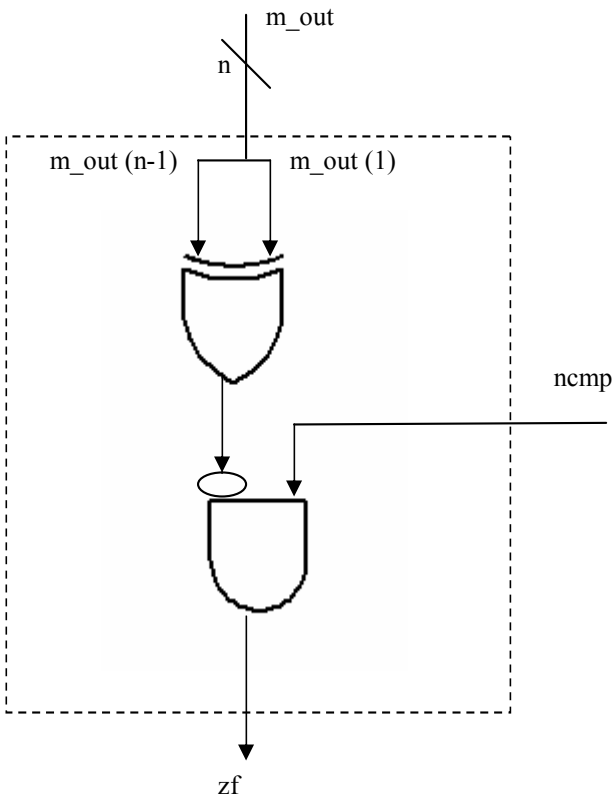
Vale 1 quando il bit meno significativo del risultato dell'ultima operazione aritmetica non contiene 1, 0 altrimenti.

Graficamente avremo:

Livello 1



Livello 2



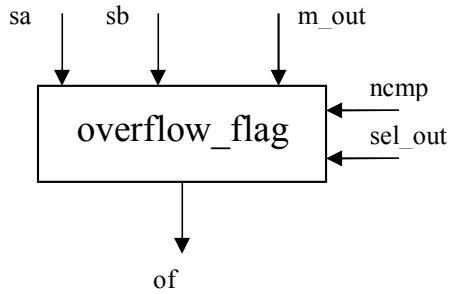
Codice di simulazione Matlab

```
%            Progetto ALU
%
%       Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%       http://www.dicecca.net

% pf assume valore 1 quando non è stato effettuato un CMP e il numero di bit
% uguali ad 1 del risultato è pari
function pf=parity_flag(ncmp,m_out)

n=length(m_out);

casc_xor= m_out(1);

for i=2:n
   casc_xor=xor(casc_xor, m_out(i));
end

pf=~casc_xor&ncmp;
```

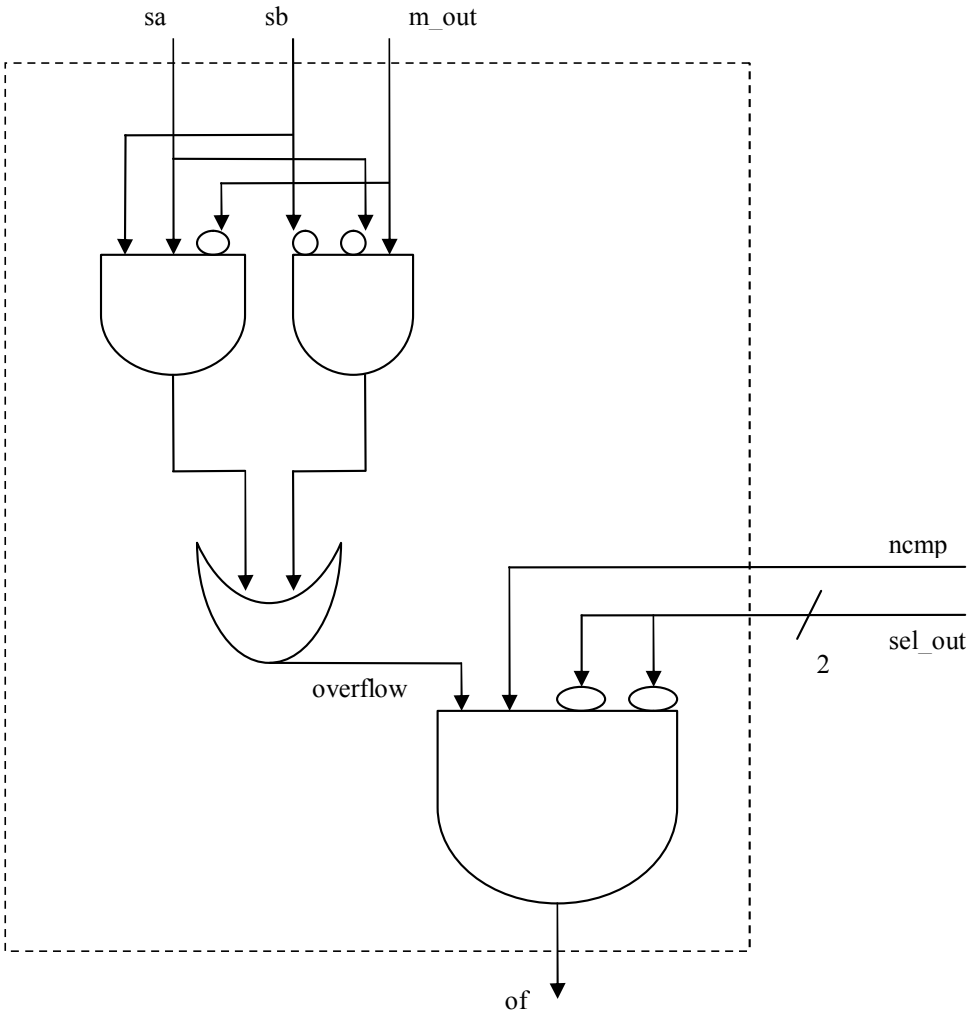
Overflow_flag

È posto ad 1 si il risultato intero è troppo grande rispetto almassimo numero positivo rappresentabile o troppo piccolo rispetto al minimo numero negativo rappresentabile. Il flagindica un overflowper numeri interi con segno rappresentati in complemento a 2.

Livello 1



Livello 2



Codice di simulazione Matlab

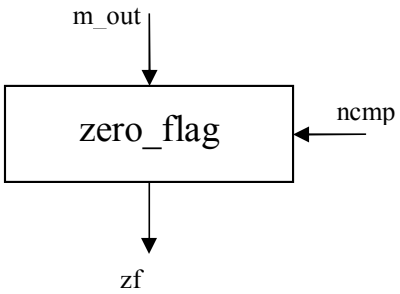
```
%             Progetto ALU
%
%       Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887            408 / 466
%
%       http://www.dicecca.net


% of assume valore 1 se il risultato di un operazione aritmetica, ad esclusione
% di CMP, da luogo ad overflow
function of=overflow_flag(ncmp,m_out,sa,sb,sel_out)

overflow=(sa&sb&~m_out)|(~sa&~sb&m_out);

of=(overflow)&(ncmp)&(~sel_out(1))&(~sel_out(2));
```

Zero_flag

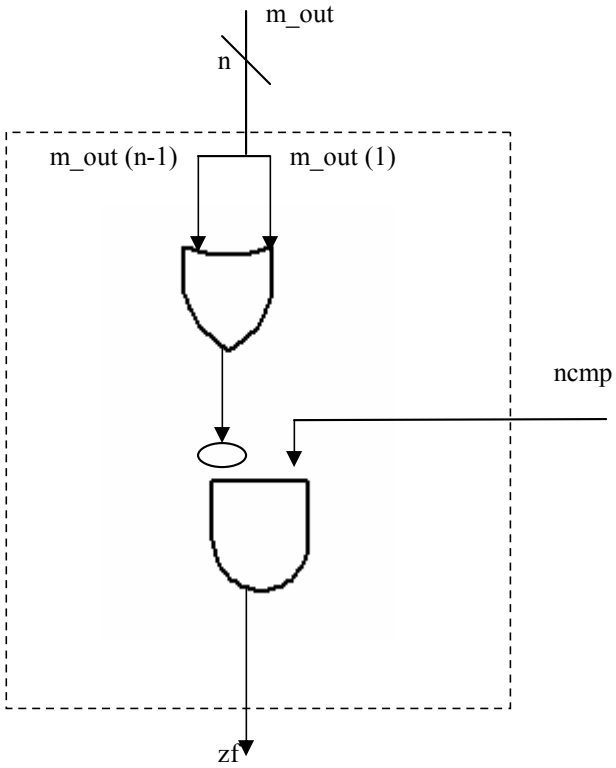
Vale 1 se il risultato dell'ultima operazione aritmetica è uguale a zero, 0 altrimenti.

Graficamente avremo:

Livello 1



Livello 2



Codice di simulazione Matlab

```
%             Progetto ALU
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%       50 / 887            408 / 466
%
%        http://www.dicecca.net


% zf da valore 1 quando, nel caso di operazione aritmetica o logica
% il risultato è 0
function zf=zero_flag(ncmp,m_out)

n=length(m_out);

casc_or= m_out(1);

for i=2:n
   cas_or=casc_or|m_out(i);
end

zf=ncmp&~casc_or;
```

Programmi che sfruttano l'ALU

Cenni preliminari

Le seguenti versioni del programma che calcola il **Teorema di Pitagora**, sono nato dall'ipotesi di poter applicare praticamente il progetto.

Naturalmente l'ALU non è una CPU in senso stretto, infatti non ha una Control Unit, ne dei registri al quale appoggiarsi per depositare i dati elaborati.

Per poter simulare una applicazione che sfruttasse l'ALU del progetto, abbiamo sfruttato un minimo di ambiente MATLAB e poche variabili, che devono essere considerati come registri temporanei all'interno della CPU.

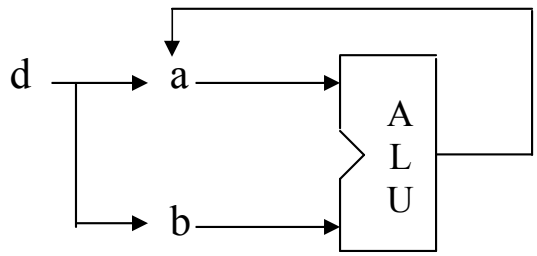
Descrizione del programma
Forma base

$$c^2 = a^2 + b^2$$

Il caso che ci siamo proposti di risolvere è la forma base del **Teorema di Pitagora: $C^2=A^2+B^2$**

Il primo problema che abbiamo analizzato è stato il calcolo del quadrato del valore inserito usando solo l'addizione.

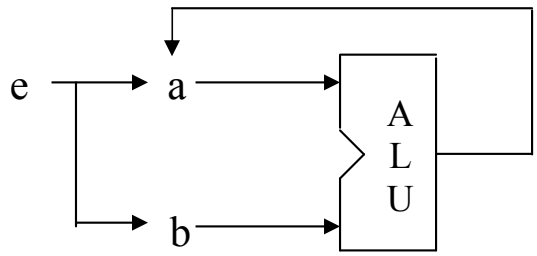
Lo schema logico è:



For r=2 to d

dove ***d*** è un valore inserito da tastiera, e a fine computazione il dato viene memorizzato in ***d***

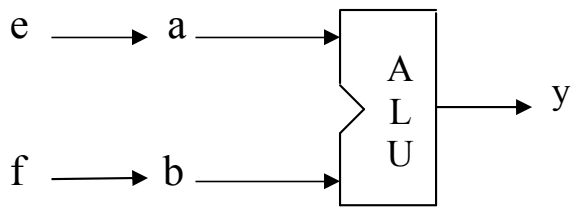
dicasi la stessa cosa per il secondo valore dell'operando



For r=2 to e

dove ***e*** è un valore inserito da tastiera, e a fine computazione il dato viene memorizzato in ***e***

Una volta finito il computo del quadrato, i risultati appoggiati nelle variabili (intese come registri) vengono a loro volta sommati:



Da intendersi come il risultato della somma dei due quadrati

Codice di simulazione Matlab

```
%            Progetto ALU
%
% Forma base del Teorema di Pitagora C²=A²+B²
%
% Valori definiti nel programma
%
%       Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%       http://www.dicecca.net

% Pulisci memoria
clear

% Pulisci schermo
clc

% Metti il valore in a il valore 3
a=[0 0 0 0 0 1 1]

% Associa a b lo stesso valore di a
b=a

% Calcola in decimale il valore binario di a
c=bin2int(a)

% Carry iniziale valido per due computazioni
c_flag=0

% Operaizone di somma valido per due computazioni
op_code=[0 0 0 0]

% Calcola il quadrato di a
for r=2 : c

    % Carica l'interfaccia dell'ALU ed esegue il calcolo
    [y,flags]=alu(a,b,op_code,c_flag)

    a=y;

end

% Stampa a video il quadrato calcolato
bin2int(y)

% Deposita il valore in d
d=y;

% Metti il valore in a il valore 4
a=[0 0 0 0 1 0 0]

% Associa a b lo stesso valore di a
b=a

% metti in c il valore decimale di a
c=bin2int(a)
```

```
% Calcola il quadrato del secondo numero
for r=2 : c

    % Carica l'interfaccia dell'ALU ed esegue il calcolo
    [y,flags]=alu(a,b,op_code,c_flag)

    % Associa alla variabile a il valore calcolato
    a=y;

end

% Stampa a video il valore calcolato
bin2int(y)

% Associa ad e il valore calcolato
e=y;

% Ricambia i valori calcolati in a et b
a=d

b=e

% Calcola la somma dei due valori calcolati
[y,flags]=alu(a,b,op_code,c_flag)

% Stampa a video il valore in decimale
bin2int(y)
```

Codice di simulazione Matlab

```
%            Progetto ALU
%
% Forma base del Teorema di Pitagora C²=A²+B²
%
% Valori definiti da tastiera
%
%       Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%       http://www.dicecca.net



% Pulisci memoria
clear

% Pulisci schermo
clc

disp(' Programma che calcola la forma base del Teorema di Pitagora')
disp(' ')
disp(' C²=A²+B²')
disp(' ')
disp('       Programma elaborato da')
disp(' ')
disp(' Giovanni DI CECCA & Virginia BELLINO')
disp('      50 / 887           408 / 466')
disp(' ')
disp('       http://www.dicecca.net')
disp(' ')
disp(' ')

% Inserimento dei valori da tastiera in decimale
d=input('Inserire il valore di a in decimale ');
e=input('Inserire il valore di b in decimale ');

% Metti il valore in a in binario a 10 bit
% (va. Min. 0 - val. max. 1023 in base 10)
a=int2bin(d,10)

% Associa a b lo stesso valore di a
b=a

% Carry iniziale valido per due computazioni
c_flag=0

% Operaizone di somma valido per due computazioni
op_code=[0 0 0 0]

% Calcola il quadrato di a
for r=2 : d

    % Carica l'interfaccia dell'ALU ed esegue il calcolo
    [y,flags]=alu(a,b,op_code,c_flag)


    % Associa ad a il valore calcolato
    a=y;
```

```
end
% Stampa a video il quadrato calcolato
bin2int(y)

% Deposita il valore in d
d=y;

% Metti il valore in a il secondo valore
a=int2bin(e,10)

% Associa a b lo stesso valore di a
b=a

% Calcola il quadrato del secondo numero
for r=2 : e

    % Carica l'interfaccia dell'ALU ed esegue il calcolo
    [y,flags]=alu(a,b,op_code,c_flag)

    a=y;

end

% Stampa a video il valore calcolato
bin2int(y)

% associa ad e il valore calcolato
e=y;

% ricambia i valori calcolati in a et b
a=d

b=e

% Calcola la somma dei due valori calcolati
[y,flags]=alu(a,b,op_code,c_flag)

% stampa a video il valore in decimale
bin2int(y)
```

Descrizione del programma
Forma completa

$$c = \sqrt{a^2 + b^2}$$

A differenza del precedente programma (di cui rimane invariata la forma), questa versione non calcola solo il "*quadrato costruito sull'ipotenusa*", ma consente di calcolare anche, e semplicemente, il lato di questo quadrato. Esegue cioè la radice quadrata del valore calcolato.

Lo schema relativo al calcolo della somma dei quadrati è uguale a quello precedentemente esposto.

Il calcolo della radice quadrata viene eseguito utilizzando la seguente regola: "dato un numero n, sommando i numeri dispari compresi tra 1 ed n, si ottenere il quadrato del valore di n".

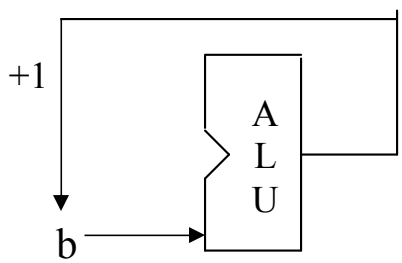
Si inserisce nel ciclo for che ha lo step di pari a r+2 (cioè somma i valori dispari al valore calcolato) un contatore che incrementa di uno il valore precedentemente calcolato.

La somma totale dei cicli effettuati fornisce esattamente la radice quadrata del numero n (nel caso si abbiano numeri rappresentanti quadrati perfetti): in caso contrario, si ottiene il valore arrotondato in eccesso.

Il codice operativo che indica di incrementare la variabile b è **op_code=[0 1 0 0]**.

Graficamente:

b=[0 0 0 0 0 0 0 0 0 0] % valore iniziale



For r=1 to c step r+2

(dove c è il valore del quadrato calcolato)

Codice di simulazione Matlab

```
%            Progetto ALU
%
% Programma per il calcolo
% del Teorema di Pitagora C=sqr(A²+B²)
%
% Valori definiti nel programma
%
%       Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%       http://www.dicecca.net


% Pulisci memoria
clear

% Pulisci schermo
clc

disp(' Programma che calcola il Teorema di Pitagora')
disp(' ')
disp(' C=sqr(A²+B²)')
disp(' ')
disp('       Programma elaborato da')
disp(' ')
disp(' Giovanni DI CECCA & Virginia BELLINO')
disp('      50 / 887           408 / 466')
disp(' ')
disp('       http://www.dicecca.net')
disp(' ')
disp(' ')

d=input('Inserire il valore di a in decimale ');
e=input('Inserire il valore di b in decimale ');

% Metti il valore in a in binario
a=int2bin(d,10)

% Associa a b lo stesso valore di a
b=a

% Carry iniziale valido per due computazioni
c_flag=0

% Operaizone di somma valido per due computazioni
op_code=[0 0 0 0]

% Calcola il quadrato di a
for r=2 : d

    % Carica l'interfaccia dell'ALU ed esegue il calcolo
    [y,flags]=alu(a,b,op_code,c_flag)

    a=y;

end
```

```
% Stampa a video il quadrato calcolato
bin2int(y)

% Deposita il valore in d
d=y;

% Metti il valore in a il secondo valore
a=int2bin(e,10)

% Associa a b lo stesso valore di a
b=a

% Calcola il quadrato del secondo numero
for r=2 : e

    % Carica l'interfaccia dell'ALU ed esegue il calcolo
    [y,flags]=alu(a,b,op_code,c_flag)

    a=y;

end

% Stampa a video il valore calcolato
bin2int(y)

% associa ad e il valore calcolato
e=y;

% ricambia i valori calcolati in a et b
a=d
b=e

% Calcola la somma dei due valori calcolati
[y,flags]=alu(a,b,op_code,c_flag)

% Stampa a video il valore in decimale e conservalo in c
c=bin2int(y)


% Inizio routine per il calcolo della radice quadrata

% Annulla la variabile
b=[0 0 0 0 0 0 0 0 0 0]

% Attiva il codice di incrementazione della varaibile b
op_code=[0 1 0 0]

% Ciclo for che calcola la successione dei numeri dispari arrotondando al
% intero superiore
for r=1 : r+2 : c

    % Carica l'interfaccia ALU
    [y,flags]=alu(a,b,op_code,c_flag)

    % Memorizza il valore calcolato in b
    b=y;
end

% Stampa il risultato
bin2int(y)
```

Multiplexer

Cenni preliminari

Il Multiplexer è un componente elettronico che permette di selezionare sull'unica uscita uno tra i diversi ingressi presenti nel sistema.

La selezione dell'ingresso che provvede a mettere a disposizione il suo contenuto sulla linea di uscita avviene mediante l'uso di apposite linee (dette linee di selezione o chipselet). Ovviamente un numero dei segnali di selezione deve essere un intero maggiore o uguale al logaritmo in base due del numero dei segnali tra cui scegliere $(\log_2[\text{controlsingal}])$.

I multiplexer che andremo ad analizzare sono:

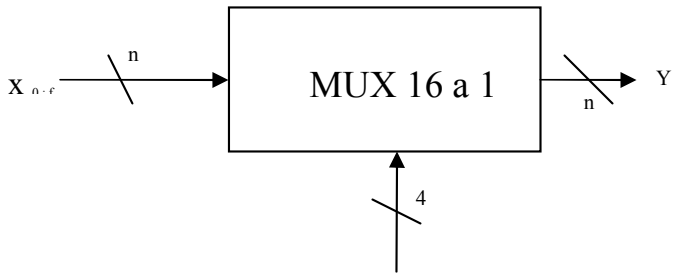
16 a 1
8 a 1
4 a 1
2 a 1

tutti a *n* bit

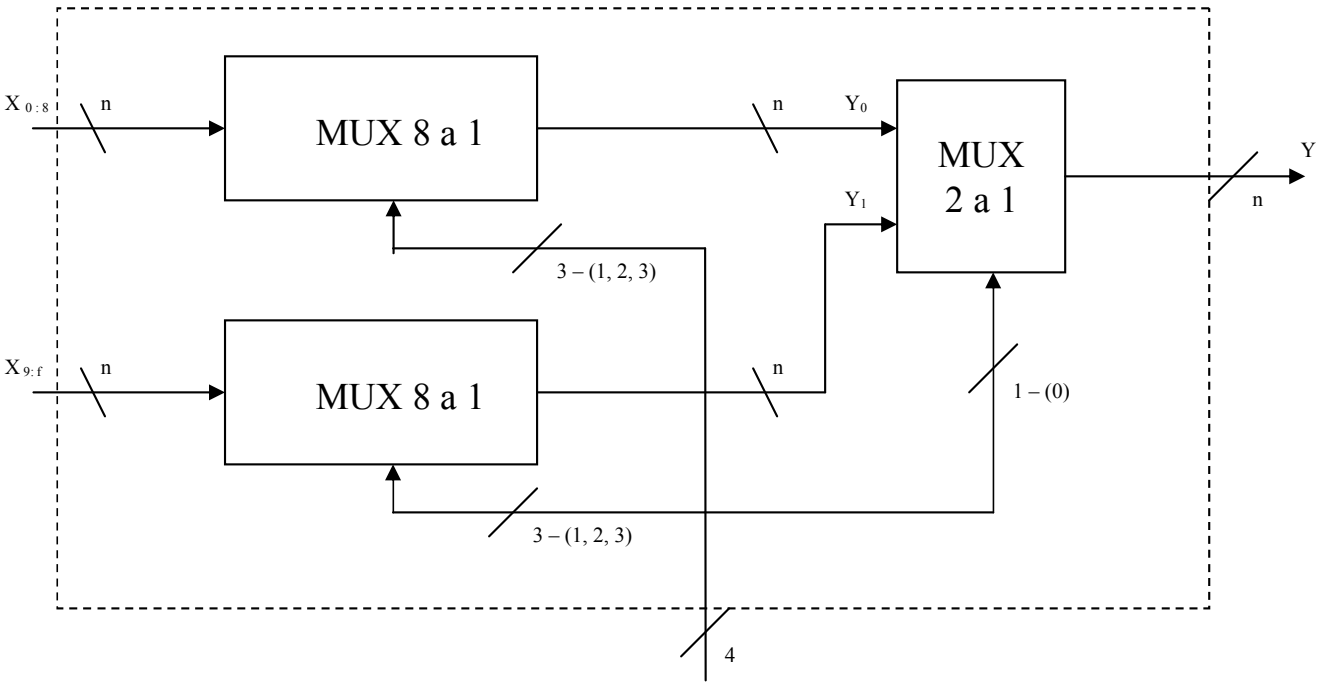
Come detto nella Introduzione il tutto verrà effettuato mediante un procedimento Top Down

Multiplexer 16 a 1

Livello 1



Livello 2



Il **Mux 16 a 1** prevede l'utilizzo di 2 **Mux 8 a 1** e di un **Mux 2 a 1**.

Il primo **Mux 8 a 1** riceve le linee che vanno da 0 a 8, mentre il secondo le linee che vanno da 9 a 15 (da 9 a f usando una codifica esadecimale).

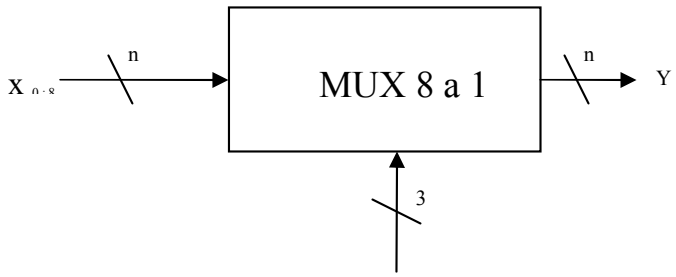
I bit di selezione da 1 a 3 vanno a fare le selezioni direttamente nei **Mux 8 a 1**, mentre il bit più significativo (il bit 0) va a fare la selezione del **Mux 2 a 1** che da poi l'uscita finale del **Mux 16 a 1**

Codice Matlab per la simulazione

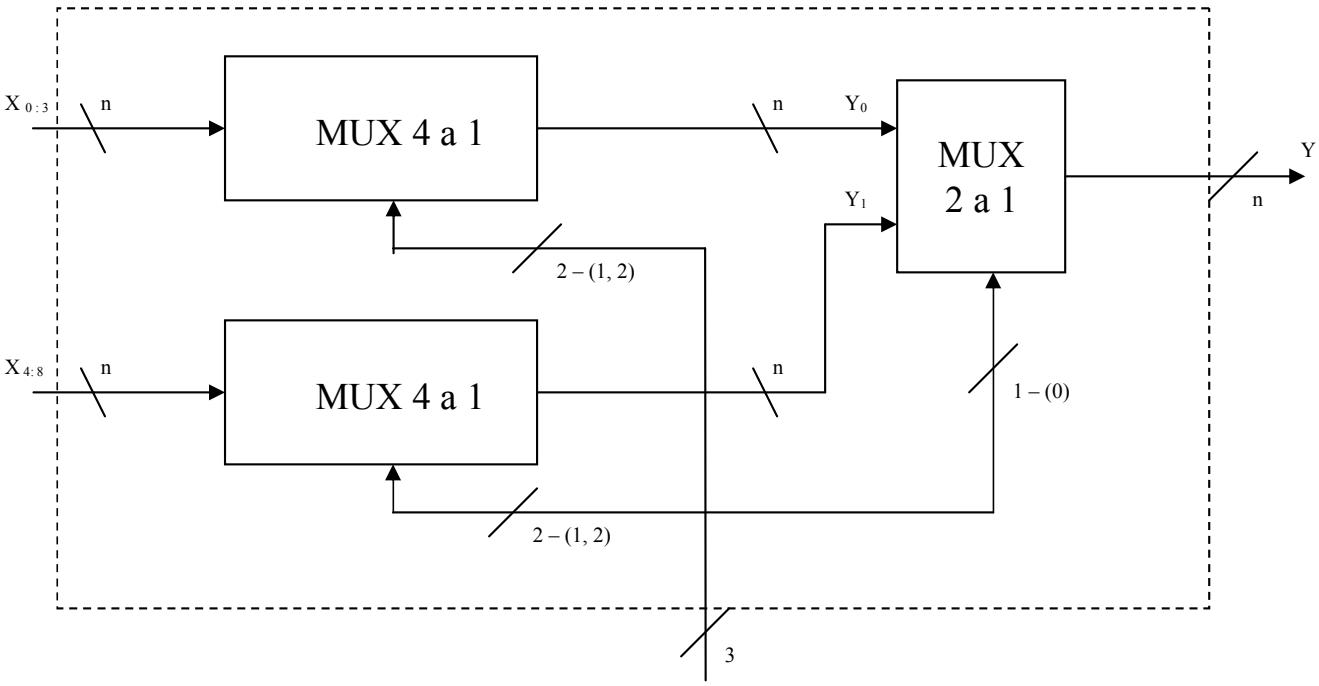
```
% IMPLEMENTAZIONE DI MUX 16:1 AD N BIT
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%         http://www.dicecca.net

% sel è un vettore binario contenente i 4 bit di selezione
% La funzione utilizza un approccio top-down scomponendo il mux 8:1 in due
% mux 8:1 e un mux 2:1


function y=muxn16_1(x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,sel);


% creazione di un array che evidenzia i tre bit di selezione meno significativi
% che gestiranno
% i due mux intermedi 8:1
sel_a=[ sel(2) sel(3) sel(4)];


% Con le due istruzioni seguenti viene effettuata una prima selezione degli
% ingressi basata
% sul valore dei bit meno significativi seguendo il medesimo approccio
% illustrato per il mux 4:1

y0=muxn8_1(x0,x1,x2,x3,x4,x5,x6,x7,sel_a);
y1=muxn8_1(x8,x9,x10,x11,x12,x13,x14,x15,sel_a);

% Selezione dell'ingresso definitivo da collegare all'uscita in base al valore
% di sel(1)[bit più :
% se sel(1)= 0 viene selezionato l'ingresso memorizzato in y0
% se sel(1)= 1 viene selezionato l'ingresso memorizzato in y1
y=muxn2_1(y0,y1,sel(1));
```

Multiplexer 8 a 1

Livello 1



Livello 2



Il **Mux 8 a 1** prevede l'utilizzo di 2 **Mux 4 a 1** e di un **Mux 2 a 1**.

Il primo **Mux 4 a 1** riceve le linee che vanno da 0 a 3, mentre il secondo le linee che vanno da 4 a 8.

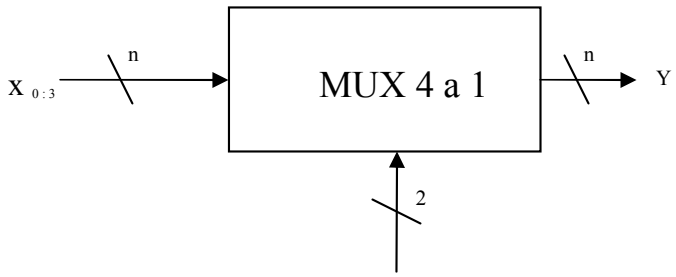
I bit di selezione 1 e 2 vanno a fare le selezioni direttamente nei **Mux 4 a 1**, mentre il bit più significativo (il bit 0) va a fare la selezione del **Mux 2 a 1** che da poi l'uscita finale del **Mux 8 a 1**

Codice Matlab per la simulazione

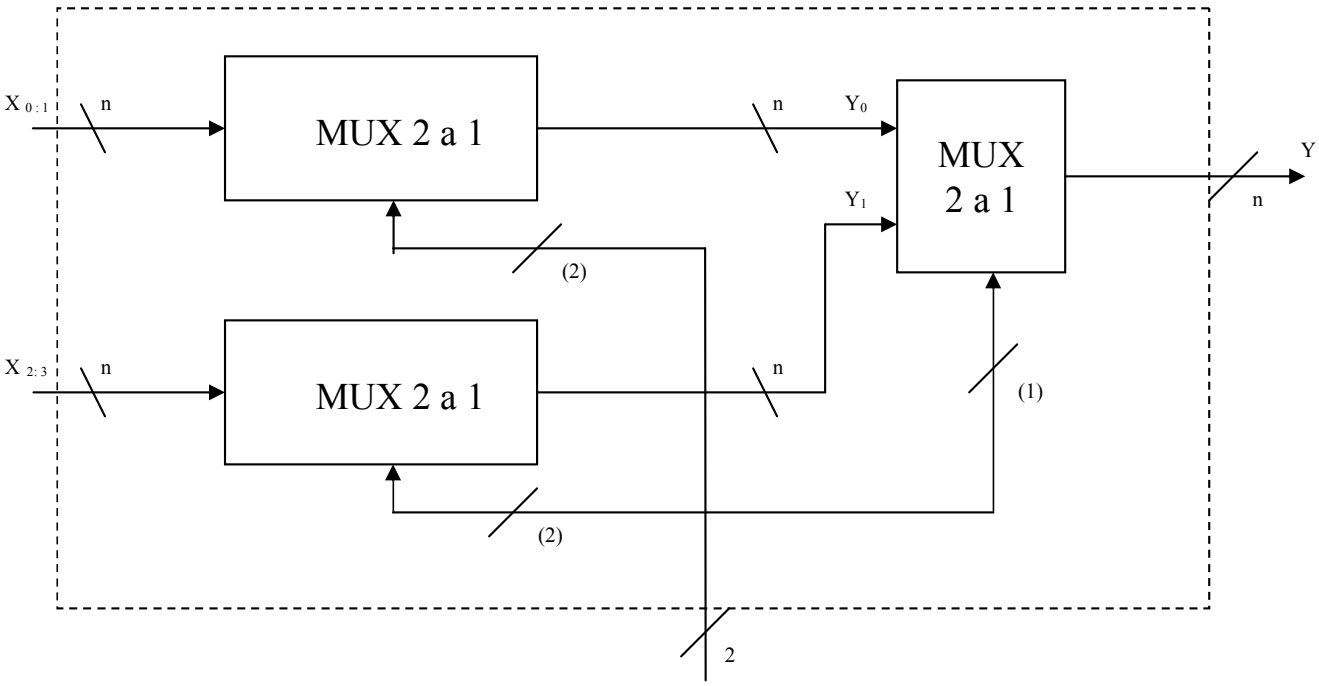
```
% Multiplexer 8:1 a n bit
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%         http://www.dicecca.net

function y=muxn8_1(x0,x1,x2,x3,x4,x5,x6,x7,sel)

% Scorporo delle condizioni di sel
sel_a=[sel(2) sel(3)];

% Carica i risultati intermedi
y0=muxn4_1(x0,x1,x2,x3,sel_a);
y1=muxn4_1(x4,x5,x6,x7,sel_a);

% Passa i risultati intermedi al Mux 2 a 1 e come chipselect
% usa il bit più significativo
y=muxn2_1(y0,y1,sel(1));
```

Multiplexer 4 a 1

Livello 1



Livello 2



Il **Mux 4 a 1** prevede l'utilizzo di 3 **Mux 2 a 1**.

Il primo **Mux 2 a 1** riceve le linee 0 et 1, mentre il secondo le linee 2 et 3.

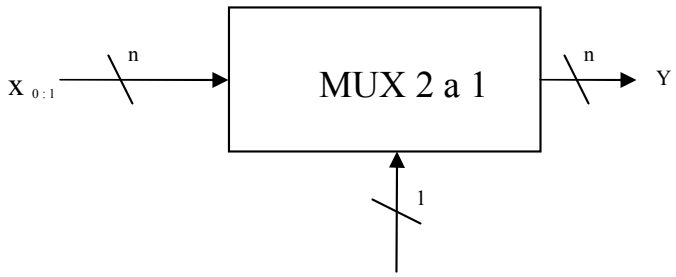
Il bit di selezione 2 va a fare la selezione direttamente nei **Mux 2 a 1**, mentre il bit più significativo (il bit 1) va a fare la selezione del **Mux 2 a 1** che da poi l'uscita finale del **Mux 4 a 1**

Codice Matlab per la simulazione

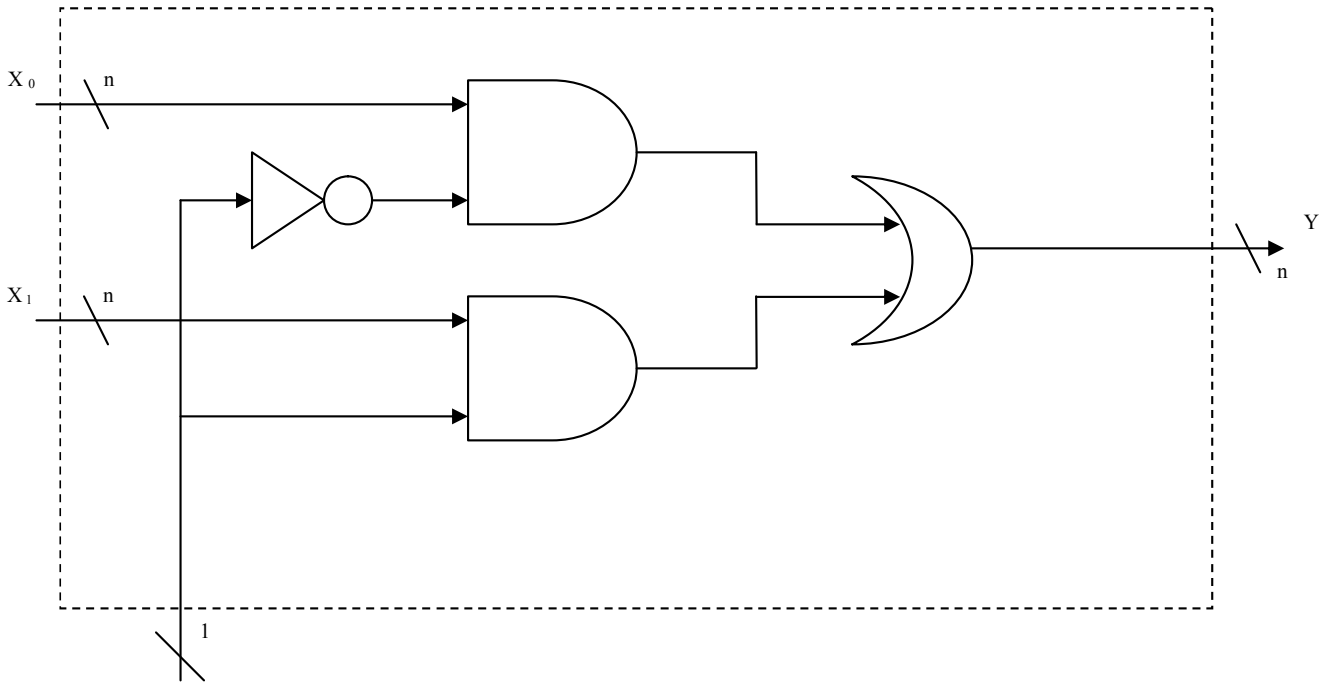
```
% IMPLEMENTAZIONE DI MUX 4:1 AD N BIT
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%         http://www.dicecca.net
% x0,x1,x2,x3 sono 4 vettori binari di n bit inseriti in ingresso
% sel è un vettore binario contenente i 2 bit di selezione
% La funzione utilizza un approccio top-down scomponendo il mux 4:1 in tre
% mux 2:1


function y=muxn4_1(x0,x1,x2,x3,sel);

% Con le due istruzioni seguenti viene effettuata una prima selezione degli
% ingressi basata
% sul valore di sel(2)[bit meno significativo]:
% se sel(2)=0 vengono selezionati x0 e x2
% se sel(2)=1 vengono selezionati x1 e x3
y0=muxn2_1(x0,x1,sel(2));
y1=muxn2_1(x2,x3,sel(2));

% Selezione dell'ingresso definitivo da collegare all'uscita in base al valore
% di sel(1):
% se sel(1)= 0 viene selezionato l'ingresso memorizzato in y0
% se sel(1)= 1 viene selezionato l'ingresso memorizzato in y1
y=muxn2_1(y0,y1,sel(1));
```

Multiplexer 2 a 1

Livello 1



Livello 2



Il **Mux 2 a 1** prevede l'utilizzo di un **Mux 2 a 1 singolo bit**.

Il bit di selezione va a fare la selezione direttamente nel **Mux 2 a 1**. Essendo il Mux a n bit, per poter gestire ogni singolo bit si serve di un ciclo for che, mediante la funzione ***lenght*** calcola di quanti bit è composta la stringa, e, la passa al ciclo for che provvede a calcolarli uno per uno.

Lo schema del Livello 2 è quello proprio di un Mux 2 a 1 a singolo bit.

Codice Matlab per la simulazione

Mux 2 a 1 a n bit

```
% Implementazione di mux 2:1 a n bit
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%         http://www.dicecca.net
% Gli argomenti x0 e x1 sono vettori binari
% y rappresenta il valore che si manifesta in uscita
% sel è il bit di selezione che determina quale ingresso si manifesta in uscita
% Per ottenere il risultato la function utilizza in cascata il mux 2:1 a bit
% singolo

function y=muxn2_1(x0,x1,sel);

sel_n=~sel;

% Definizione del contatore n che risulta pari alla lunghezza del primo vettore
% inserito
n=length(x0);

% Determinazione del valore da manifestare in uscita
for i=1:n

% L'istruzione successiva definisce due possibilità:
% 1-se sel=1 e sel_n=0 si manifesta in uscita il valore di x1
% 1-se sel=0 e sel_n=1 si manifesta in uscita il valore di x0

   y(i)=mux2_1(x0(i),x1(i),sel,sel_n);

end % End del for per i bit che fanno parte dell'array
```

Mux 2 a 1 a singolo bit

```
% Implementazione di un mux2:1 a singolo bit
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%         http://www.dicecca.net
% x0 e x1 sono linee dati(ingressi) a singolo bit
% sel e sel_n sono le linee di selezione a singolo bit in forma vera e negata
% y rappresenta il valore che si manifesta in uscita

function y=mux2_1(x0,x1,sel,sel_n)

%La seguente istruzione prevede due possibilità:
%1-se sel_n=1 viene selezionato in output il valore del primo ingresso
%2-se sel=1 viene selezionato in output il valore del secondo ingresso

y=(x0&sel_n)|(x1&sel);
```

Esempi d'uso

L'esempio d'uso è importante perché consente di analizzare il corretto funzionamento degli script (così in Matlab vengono chiamate le funzioni).

Per fare ciò abbiamo utilizzato uno script che consente di fare ciò.

Test del MUX 16 a 1

```
% Test del Mux 16 a 1
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%         http://www.dicecca.net

% Pulisci memoria
clear

% Pulisci schermo
clc

% Inserisici i valori nel vettore
x0=[0 0 0 0]
x1=[0 0 0 1]
x2=[0 0 1 0]
x3=[0 0 1 1]
x4=[0 1 0 0]
x5=[0 1 0 1]
x6=[0 1 1 0]
x7=[0 1 1 1]
x8=[1 0 0 0]
x9=[1 0 0 1]
x10=[1 0 1 0]
x11=[1 0 1 1]
x12=[1 1 0 0]
x13=[1 1 0 1]
x14=[1 1 1 0]
x15=[1 1 1 1]

% Valori del Chipselect che è uguale al valore di x13
sel=[1 1 0 1]

% Stampa il risultato
disp('Stampa il risultato')

% Carica il MUX 16 a 1
y=muxn16_1(x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,sel)
```

```
Stampa il risultato

y =

     1     1     0     1

EDU>>
```

Test del MUX 8 a 1

```
% Test del Mux 8 a 1
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%         http://www.dicecca.net

% Pulisci memoria
clear

% Pulisci schermo
clc

% Inserisici i valori nel vettore
x0=[0 0 0 0]
x1=[0 0 0 1]
x2=[0 0 1 0]
x3=[0 0 1 1]
x4=[0 1 0 0]
x5=[0 1 0 1]
x6=[0 1 1 0]
x7=[0 1 1 1]


% Valori del Chipselect che è uguale al valore di x5
sel=[1 0 1]

% Stampa il risultato
disp('Stampa il risultato')

% Carica il MUX 8 a 1
y=muxn8_1(x0,x1,x2,x3,x4,x5,x6,x7,sel)
```

```
Stampa il risultato

y =

     0     1     0     1

EDU>>
```

Test del MUX 4 a 1

```
% Test del Mux 4 a 1
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887            408 / 466
%
%         http://www.dicecca.net


% Pulisci memoria
clear

% Pulisci schermo
clc

% Inserisici i valori nel vettore
x0=[0 0 0 0]
x1=[0 0 0 1]
x2=[0 0 1 0]
x3=[0 0 1 1]


% Valori del Chipselect che è uguale al valore di x3
sel=[1 1]

% Stampa il risultato
disp('Stampa il risultato')

% Carica il MUX 4 a 1
y=muxn4_1(x0,x1,x2,x3,sel)
```

```
Stampa il risultato

y =

    0    0    1    1

EDU>>
```

Test del MUX 2 a 1

```
% Test del Mux 2 a 1
%
%        Programma elaborato da
%
% Giovanni DI CECCA & Virginia BELLINO
%      50 / 887           408 / 466
%
%         http://www.dicecca.net

% Pulisci memoria
clear

% Pulisci schermo
clc

% Inserisici i valori nel vettore
x0=[0 0 0 0]
x1=[0 0 0 1]


% Valori del Chipselect che è uguale al valore di x3
sel=[1]

% Stampa il risultato
disp('Stampa il risultato')

% Carica il MUX 2 a 1
y=muxn2_1(x0,x1,sel)
```

```
Stampa il risultato

y =

    0    0    0    1

EDU>>
```

Appendice

In questa appendice sono riportati i tabulati di tutte le prove effettuate sia con l'ALU che con i Multiplexer direttamente con MATLAB.

```
a =

     0     0     0     0     0     1     1

b =

     0     0     0     0     0     1     1

c =

     3

c_flag =

     0

op_code =

     0     0     0     0

y =

     0     0     0     0     1     1     0

flags =

     0     0     1     0     1

y =

     0     0     0     1     0     0     1

flags =

     0     0     1     0     1

ans =

     9

a =

     0     0     0     0     1     0     0

b =

     0     0     0     0     1     0     0

c =
```

```
     4

y =

     0     0     0     1     0     0     0

flags =

     0     0     0     0     1

y =

     0     0     0     1     1     0     0

flags =

     0     0     1     0     1

y =

     0     0     1     0     0     0     0

flags =

     0     0     0     0     1

ans =

    16

a =

     0     0     0     1     0     0     1

b =

     0     0     1     0     0     0     0

y =

     0     0     1     1     0     0     1

flags =

     0     0     0     0     1

ans =

    25
```

```
 Programma che calcola la forma base del Teorema di Pitagora

 C²=A²+B²

       Programma elaborato da

 Giovanni DI CECCA & Virginia BELLINO
      50 / 887           408 / 466

       http://www.dicecca.net


Inserire il valore di a in decimale 12
Inserire il valore di b in decimale 16

a =

     0     0     0     0     0     0     1     1     0     0


b =

     0     0     0     0     0     0     1     1     0     0


c_flag =

     0


op_code =

     0     0     0     0


y =

     0     0     0     0     0     1     1     0     0     0


flags =

     0     0     1     0     1


y =

     0     0     0     0     1     0     0     1     0     0


flags =

     0     0     1     0     1


y =

     0     0     0     0     1     1     0     0     0     0


flags =
```

```
     0     0     1     0     1

y =

     0     0     0     0     1     1     1     1     0     0

flags =

     0     0     1     0     1

y =

     0     0     0     1     0     0     1     0     0     0

flags =

     0     0     1     0     1

y =

     0     0     0     1     0     1     0     1     0     0

flags =

     0     0     0     0     1

y =

     0     0     0     1     1     0     0     0     0     0

flags =

     0     0     1     0     1

y =

     0     0     0     1     1     0     1     1     0     0

flags =

     0     0     1     0     1

y =

     0     0     0     1     1     1     1     0     0     0

flags =

     0     0     1     0     1
```

```
y =

     0     0     1     0     0     0     0     1     0     0

flags =

     0     0     1     0     1

y =

     0     0     1     0     0     1     0     0     0     0

flags =

     0     0     1     0     1

ans =

   144

a =

     0     0     0     0     0     1     0     0     0     0

b =

     0     0     0     0     0     1     0     0     0     0

y =

     0     0     0     0     1     0     0     0     0     0

flags =

     0     0     0     0     1

y =

     0     0     0     0     1     1     0     0     0     0

flags =

     0     0     1     0     1

y =

     0     0     0     1     0     0     0     0     0     0

flags =
```

```
     0     0     0     0     1

y =

     0     0     0     1     0     1     0     0     0     0

flags =

     0     0     1     0     1

y =

     0     0     0     1     1     0     0     0     0     0

flags =

     0     0     1     0     1

y =

     0     0     0     1     1     1     0     0     0     0

flags =

     0     0     0     0     1

y =

     0     0     1     0     0     0     0     0     0     0

flags =

     0     0     0     0     1

y =

     0     0     1     0     0     1     0     0     0     0

flags =

     0     0     1     0     1

y =

     0     0     1     0     1     0     0     0     0     0

flags =

     0     0     1     0     1
```

```
y =

     0     0     1     0     1     1     0     0     0     0

flags =

     0     0     0     0     1

y =

     0     0     1     1     0     0     0     0     0     0

flags =

     0     0     1     0     1

y =

     0     0     1     1     0     1     0     0     0     0

flags =

     0     0     0     0     1

y =

     0     0     1     1     1     0     0     0     0     0

flags =

     0     0     0     0     1

y =

     0     0     1     1     1     1     0     0     0     0

flags =

     0     0     1     0     1

y =

     0     1     0     0     0     0     0     0     0     0

flags =

     0     0     0     0     1
```

```
ans =

   256


a =

     0     0     1     0     0     1     0     0     0     0


b =

     0     1     0     0     0     0     0     0     0     0


y =

     0     1     1     0     0     1     0     0     0     0


flags =

     0     0     0     0     1


ans =

   400

EDU>>
```

```
 Programma che calcola il Teorema di Pitagora

 C=sqr(A²+B²)

       Programma elaborato da

 Giovanni DI CECCA & Virginia BELLINO
      50 / 887           408 / 466

       http://www.dicecca.net


Inserire il valore di a in decimale 3
Inserire il valore di b in decimale 4

a =

     0     0     0     0     0     0     0     0     1     1


b =

     0     0     0     0     0     0     0     0     1     1


c_flag =

     0


op_code =

     0     0     0     0


y =

     0     0     0     0     0     0     0     1     1     0


flags =

     0     0     1     0     1


y =

     0     0     0     0     0     0     1     0     0     1


flags =

     0     0     1     0     1


ans =

     9


a =
```

```
     0     0     0     0     0     0     0     1     0     0

b =

     0     0     0     0     0     0     0     1     0     0

y =

     0     0     0     0     0     0     1     0     0     0

flags =

     0     0     0     0     1

y =

     0     0     0     0     0     0     1     1     0     0

flags =

     0     0     1     0     1

y =

     0     0     0     0     0     1     0     0     0     0

flags =

     0     0     0     0     1

ans =

    16

a =

     0     0     0     0     0     0     1     0     0     1

b =

     0     0     0     0     0     1     0     0     0     0

y =

     0     0     0     0     0     1     1     0     0     1

flags =

     0     0     0     0     1
```

```
c =

    25

b =

     0     0     0     0     0     0     0     0     0     0

op_code =

     0     1     0     0

y =

     0     0     0     0     0     0     0     0     0     1

flags =

     0     0     0     0     1

y =

     0     0     0     0     0     0     0     0     1     0

flags =

     0     0     0     0     1

y =

     0     0     0     0     0     0     0     0     1     1

flags =

     0     0     1     0     1

y =

     0     0     0     0     0     0     0     1     0     0

flags =

     0     0     0     0     1

y =

     0     0     0     0     0     0     0     1     0     1

flags =
```

```
     0     0     1     0     1


ans =

     5

EDU>>
```

Test del Mux 16 a 1

Programma elaborato da

Giovanni DI CECCA & Virginia BELLINO
50 / 887 408 / 466

http://www.dicecca.net

```
x0 =

     0     0     0     0


x1 =

     0     0     0     1


x2 =

     0     0     1     0


x3 =

     0     0     1     1


x4 =

     0     1     0     0


x5 =

     0     1     0     1


x6 =

     0     1     1     0


x7 =

     0     1     1     1


x8 =

     1     0     0     0


x9 =

     1     0     0     1


x10 =

     1     0     1     0
```

```
x11 =

     1     0     1     1


x12 =

     1     1     0     0


x13 =

     1     1     0     1


x14 =

     1     1     1     0


x15 =

     1     1     1     1


sel =

     1     1     0     1

Stampa il risultato

y =

     1     1     0     1

EDU>>
```

```
          Test del Mux 8 a 1

        Programma elaborato da

Giovanni DI CECCA & Virginia BELLINO
      50 / 887           408 / 466

         http://www.dicecca.net

x0 =

     0     0     0     0


x1 =

     0     0     0     1


x2 =

     0     0     1     0


x3 =

     0     0     1     1


x4 =

     0     1     0     0


x5 =

     0     1     0     1


x6 =

     0     1     1     0


x7 =

     0     1     1     1


sel =

     1     0     1

Stampa il risultato

y =

     0     1     0     1

EDU>>
```

```
          Test del Mux 4 a 1

        Programma elaborato da

Giovanni DI CECCA & Virginia BELLINO
      50 / 887           408 / 466

         http://www.dicecca.net

x0 =

     0     0     0     0


x1 =

     0     0     0     1


x2 =

     0     0     1     0


x3 =

     0     0     1     1


sel =

     1     1

Stampa il risultato

y =

     0     0     1     1

EDU>>
```

```
         Test del Mux 2 a 1

        Programma elaborato da

Giovanni DI CECCA & Virginia BELLINO
      50 / 887           408 / 466

         http://www.dicecca.net

x0 =

     0     0     0     0


x1 =

     0     0     0     1


sel =

     1

Stampa il risultato

y =

     0     0     0     1

EDU>>
```