LEZIONE N° 91

Introduzione agli elementi architetturali principali

FORME STANDARD DI FUNZIONI BOOLEANE

Dalle tabelle di verita' alla sintesi della rete logica

- Una funzione booleana puo' essere definita tramite tabella di verita'
  - Esempio

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

- E' possibile derivare una espressione algebrica a partire dalla tabella di verita' in (almeno) due modi:
  - Identificando tutte le righe che producono un 1 e sommando (OR) i corrispondenti termini (prodotti ovvero AND di variabili) che producono tale 1. Tali «termini prodotto» prendono il nome di mintermini. L'espressione risultante e' detta somma di prodotti. Esempio:

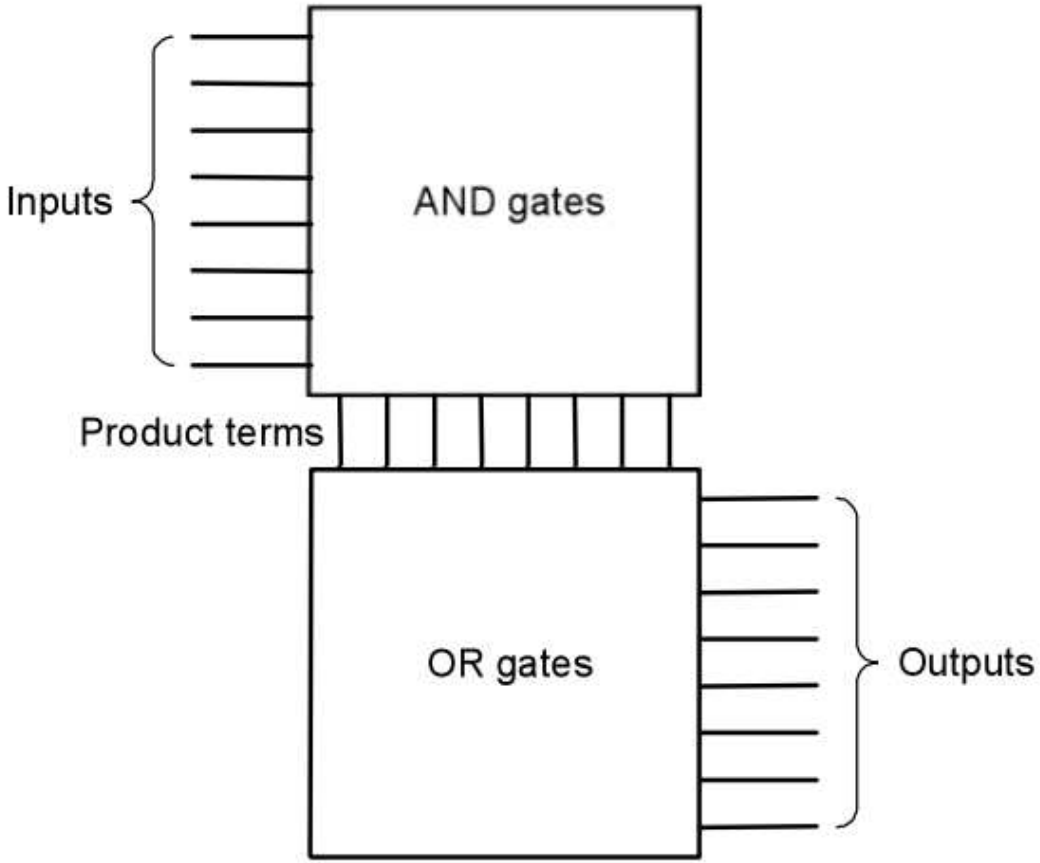
$$z = \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c}$$

  - Identificando tutte le righe che producono uno 0 e facendo il prodotto (AND) dei corrispondenti termini (somme ovvero OR di variabili) che producono tale 0. Tali «termini somma» prendono il nome di maxtermini. L'espressione risultante e' detta prodotto di somme. Esempio:

$$z = \left(a + b + \bar{c}\right) \cdot \left(a + \bar{b} + \bar{c}\right) \cdot \left(\bar{a} + \bar{b} + \bar{c}\right)$$

Programmable Logic Array - PLA

- Dato che si puo' sempre rappresentare una funzione logica come tabella di verita', si puo' sempre rappresentare anche come somma di prodotti (o prodotto di somme)
- Questo conduce ad una tecnica di implementazione universale nota come PLA

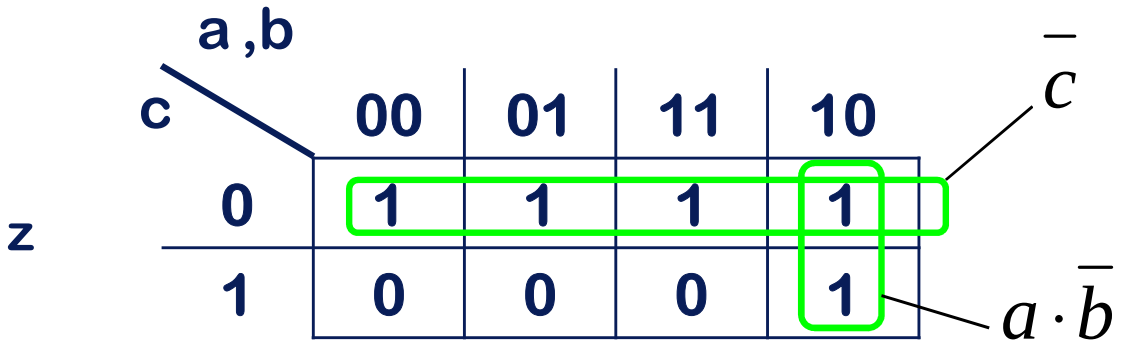


Le due matrici AND e OR possono essere anche rese "programmabili" (vedere lezioni successive)

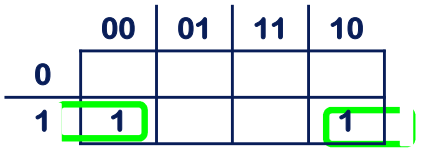
Mappe di Karnaugh

- Puo' essere comodo semplificare l'espressione booleana
  - Per ottenere espressioni booleane piu' semplici e facili da verificare
  - Per sintetizzare una rete logica con un minor numero di porte (minor area)
- La Mappa di Karnaugh e' una rappresentazione alternativa della tabella di verita' che, sfruttando la proprieta' di assorbimento*, permette rapidamente di identificare termini piu' semplici
- La Mappa di Karnaugh si ottiene:
  - disponendo gli "indici" generati dalle variabili di ingresso in modo che due indici adiacenti differiscano per una sola cifra. Esempio:

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



Nota: la mappa si svolge in realta' su una superfice toroidale → I sottocubi possono continuare su celle adiacienti la parte opposta



- Localizzando i gruppi adiacenti di 1 con un numero di 1 pari a una potenza di 2 (detti sottocubi di ordine k se contengono 2^k uni)
- Un termine e' identificato dagli indici che rimangono "fissi" nel sottocubo. Esempio:

$$z = \bar{c} + a \cdot \bar{b}$$

* $xy+x\bar{y}=x$

Mappe di Karnaugh a 2, 3, 4, 5 variabili

	0	1
0	1	1
1		

2 variabili

	00	01	11	10
0				
1		1	1	

3 variabili

	00	01	11	10
00				
01		1	1	
11		1	1	
10				

4 variabili

1

	00	01	11	10
00	1			
01				
11				
10				

0

	00	01	11	10
00	1			
01				
11				
10				

5 variabili (richiede una visione “multidimensionale”)

Note sulle Mappe di Karnaugh

- Se la mappa contiene dei non-specificato (X o -)
  - Si puo' interpretare come e' piu' conveniente per ottenere sottocubi piu' grandi
- Se ci sono piu' possibilita' con sottocubi piu' piccoli e piu' grandi
  - Si preferiscono sempre coperture con sottocubi piu' grandi (hanno meno ingressi)
- Si prendono solo i termini strettamente necessari per ottenere una lista di copertura "a costo minimo"

Karnaugh, M. "A Map method for Synthesis of Combinational Logic",
Transactions of AIEE, Communication and Electronics, 72, part I, Nov. 1953, pp. 593-99

RETI COMBINATORIE NOTEVOLI

Decoder

Encoder

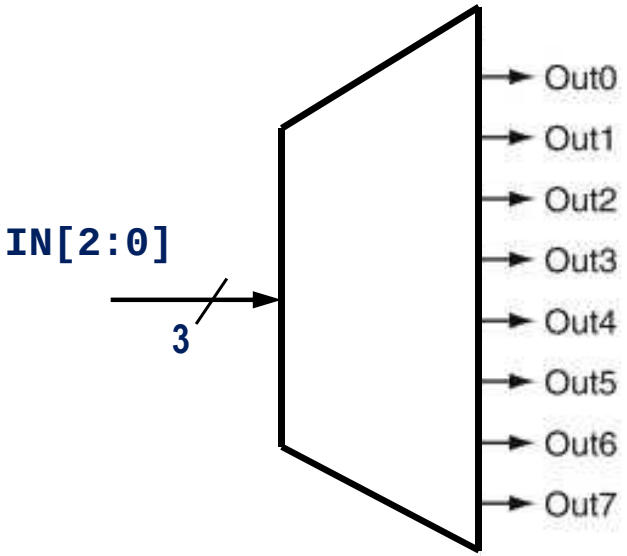
Encoder con priorita'

Multiplexer

Demultiplexer

Look-Up-Table (LUT)

Decoder



a. A 3-bit decoder

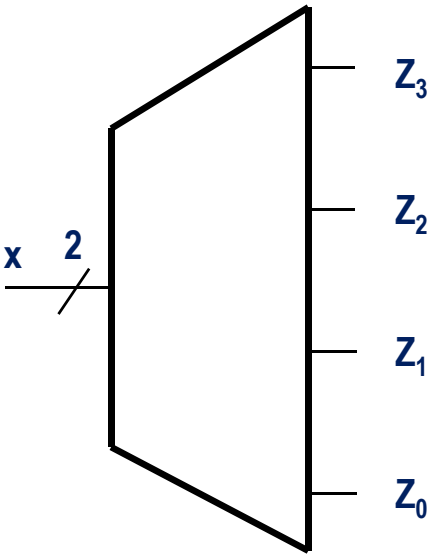
Inputs			Outputs							
IN2	IN1	IN0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

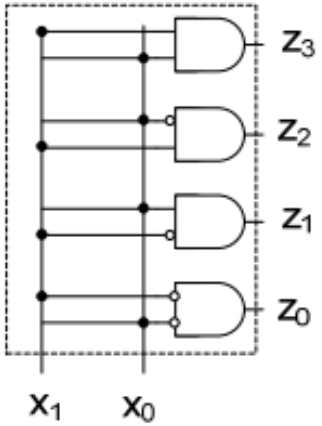
b. The truth table for a 3-bit decoder

- Esempio di decoder a 3 bit o "da-3-a-8" (v. PH5, appendice B.3)
- Un solo filo di uscita vale 1 (gli altri valgono tutti 0): quello di indice pari all'intero "posto sui fili" di ingresso
- In generale, un decoder da-n-a-2^n ha n ingressi e 2^n uscite

Realizzazione del decoder

- Si puo' sintetizzare in termini di porte logiche con 2^n porte AND ognuna delle quali riconosce uno dei 2^n mintermini





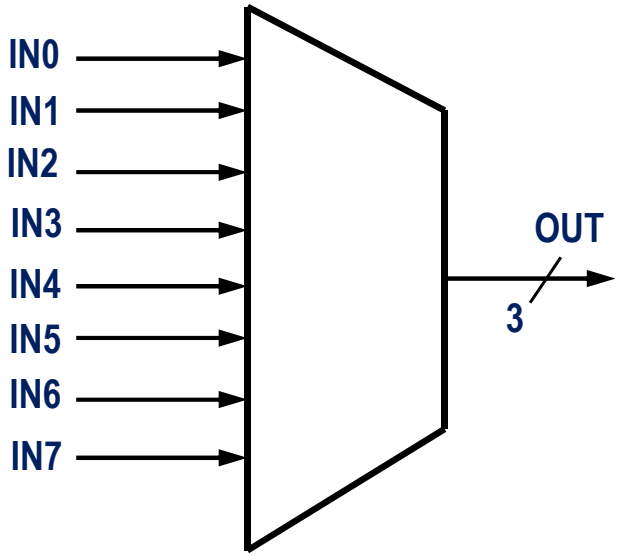
Es. decoder da-2-a-4

Nota: in realta' possono esistere implementazioni fisiche CMOS diverse e' piu' efficienti dell'elemento architetturale "decoder"

Nota2: puo' anche avere un segnale di abilitazione; si realizza mettendo ogni uscita in AND con tale segnale di abilitazione

Encoder

- L'encoder effettua l'operazione inversa del decoder
- Posto un 1 sul k-esimo dei 2^n ingressi (gli altri sono posti a 0): le n uscite producono i bit corrispondenti alla rappresentazione in base due di k



IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0	OUT2	OUT1	OUT0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Nota: questa NON E' una tabella della verita'; la si puo' interpretare come tabella di verita' immaginando che le mancanti 56 righe le uscite sono indeterminate (X o "don't care")

Realizzazione dell'encoder

- OUT0=IN1+IN3+IN5+IN7
- OUT1=IN2+IN3+IN6+IN7
- OUT2=IN4+IN5+IN6+IN7

- Dunque tale encoder puo' essere realizzato con 3 porte OR a 4 ingressi

Esercizio per casa: ricavare queste equazioni dalle mappe di Karnaugh

Priority Encoder

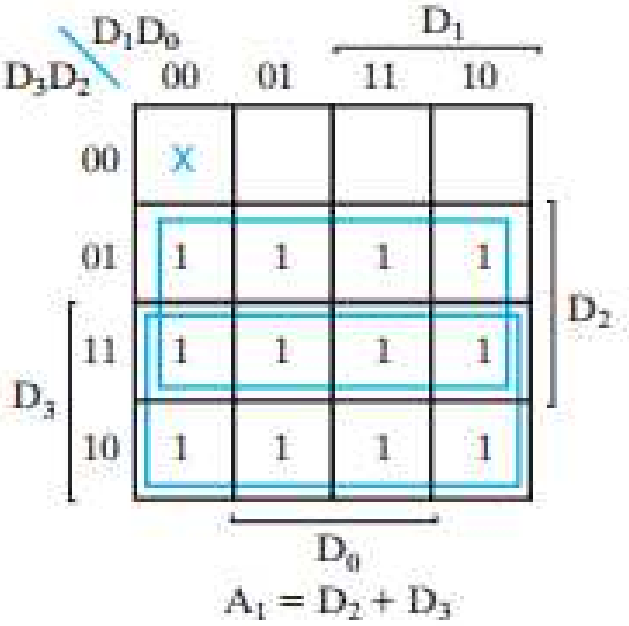
- Questa e' una rete diversa dall'encoder semplice

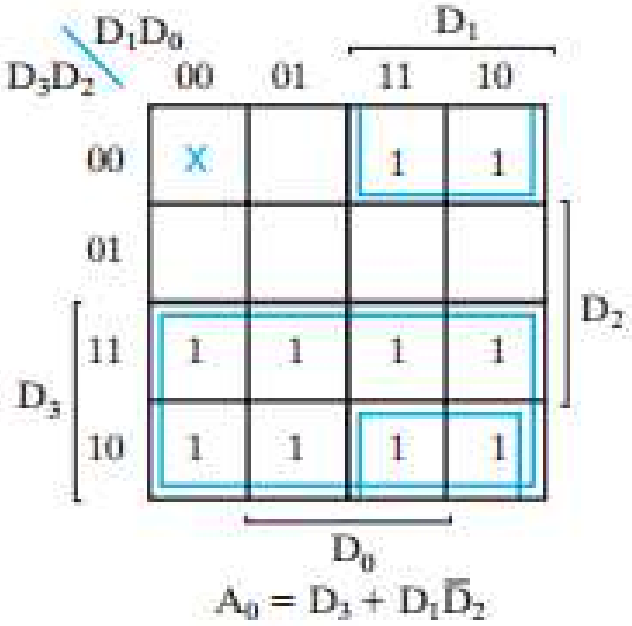
Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Tabella di verita' completa

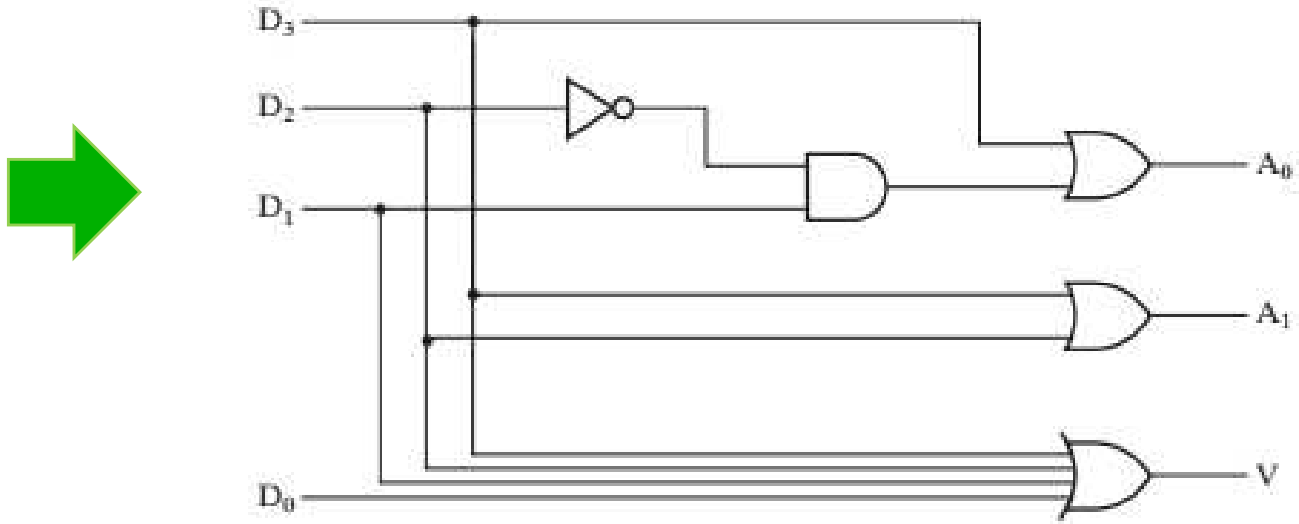
- Quando D3 vale 1 tutti gli altri bit non contano e l'uscita indica la massima priorita $(11)_2$ ovvero 3
- Quando D3 vale 0 ma D2 vale 1 gli altri bit a destra non contano e l'uscita indica priorita' 2 e cosi' via…
- Il bit V e' necessario per indicare se l'uscita e' valida ovvero se almeno un 1 e' stato posto in ingresso

Realizzazione del Priority Encoder



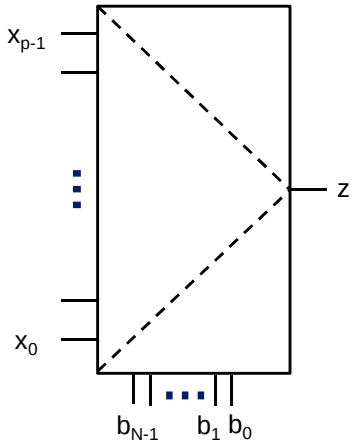


$$A_0 = D_3 + D_1\overline{D}_2$$
$$A_1 = D_2 + D_3$$
$$V = D_0 + D_1 + D_2 + D_3$$



Multiplexer

- Il multiplexer serve per selezionare un'informazione



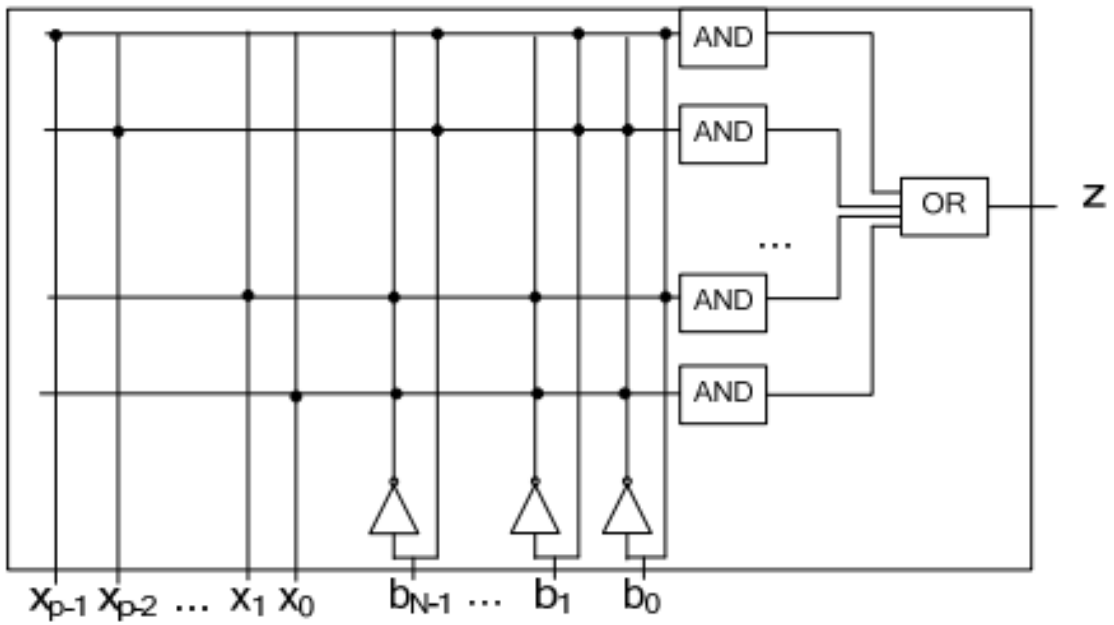
Multiplexer con 2^N ingressi

- Ha 2^N+N ingressi e 1 uscita
- Gli N ingressi b sono detti variabili di comando
- Specificando B=i viene selezionato uno (quello con indice i) dei $2^N=p$ ingressi x

$$z = x_i \Leftrightarrow (b_{N-1} \ldots b_1 b_0)_2 \equiv i$$

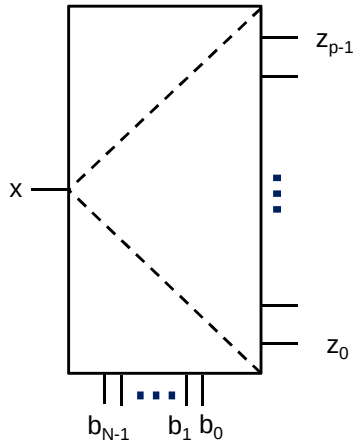
AND con N+1 ingressi

$$z = x_0 \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \ldots \cdot \overline{b_1} \cdot \overline{b_0} +$$
$$x_1 \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \ldots \cdot \overline{b_1} \cdot b_0 +$$
$$\ldots$$
$$x_{p-2} \cdot b_{N-1} \cdot b_{N-2} \cdot \ldots \cdot b_1 \cdot \overline{b_0} +$$
$$x_{p-1} \cdot b_{N-1} \cdot b_{N-2} \cdot \ldots \cdot b_1 \cdot b_0$$



Demultiplexer

- Realizza la funzione inversa del multiplexer, ovvero data un'informazione x, la distribuisce su una di 2^N uscite possibili



Demultiplexer con 2^N uscite

- Ha 1+N ingressi e 2^N uscite
- Gli N ingressi b sono detti variabili di comando
- Specificando B=i viene inviato l'ingresso x su una (quella con indice i) delle 2^N=p uscite z
- Le altre uscite valgono 0

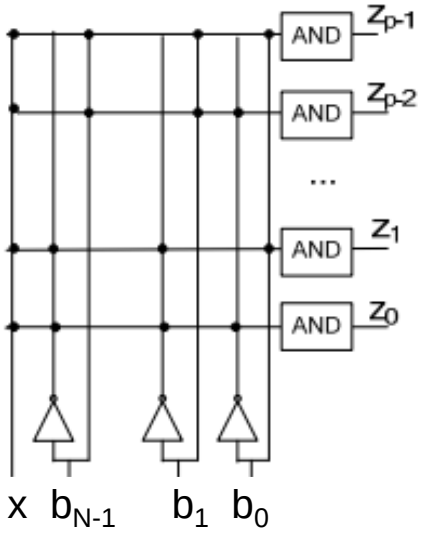
$$z_0 = x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \ldots \cdot \overline{b_1} \cdot \overline{b_0}$$

$$z_1 = x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \ldots \cdot \overline{b_1} \cdot b_0$$

...

$$z_{p-2} = x \cdot b_{N-1} \cdot b_{N-2} \cdot \ldots \cdot b_1 \cdot \overline{b_0}$$

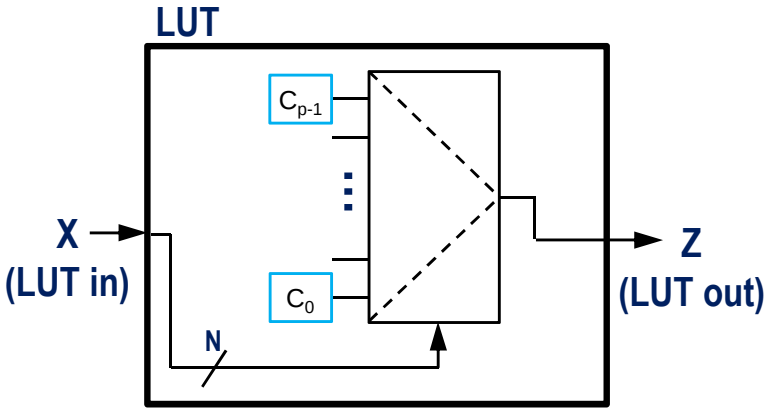
$$z_{p-1} = x \cdot b_{N-1} \cdot b_{N-2} \cdot \ldots \cdot b_1 \cdot b_0$$



Coincide col decoder con abilitazione

Look Up Table (LUT)

- E' un Multiplexer in cui gli ingressi sono costanti



- Ha 2^N+N ingressi e 1 uscita
- Gli N ingressi X fungono da variabili di comando
- Specificando X=i viene selezionato uno (quello con indice i) dei 2^N=p ingressi (FISSI)

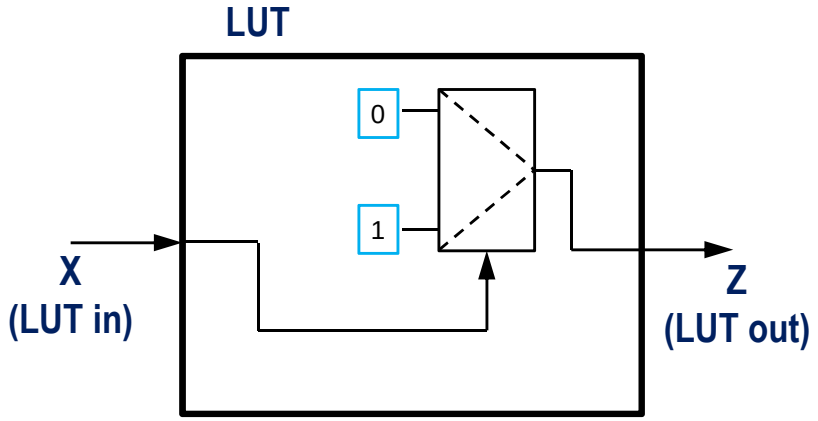
- E' utile per realizzare una funzione booleana generica
- Esempio realizzazione della funzione "inverter" tramite LUT

X	Z
0	1
1	0

Profondita' (depth)

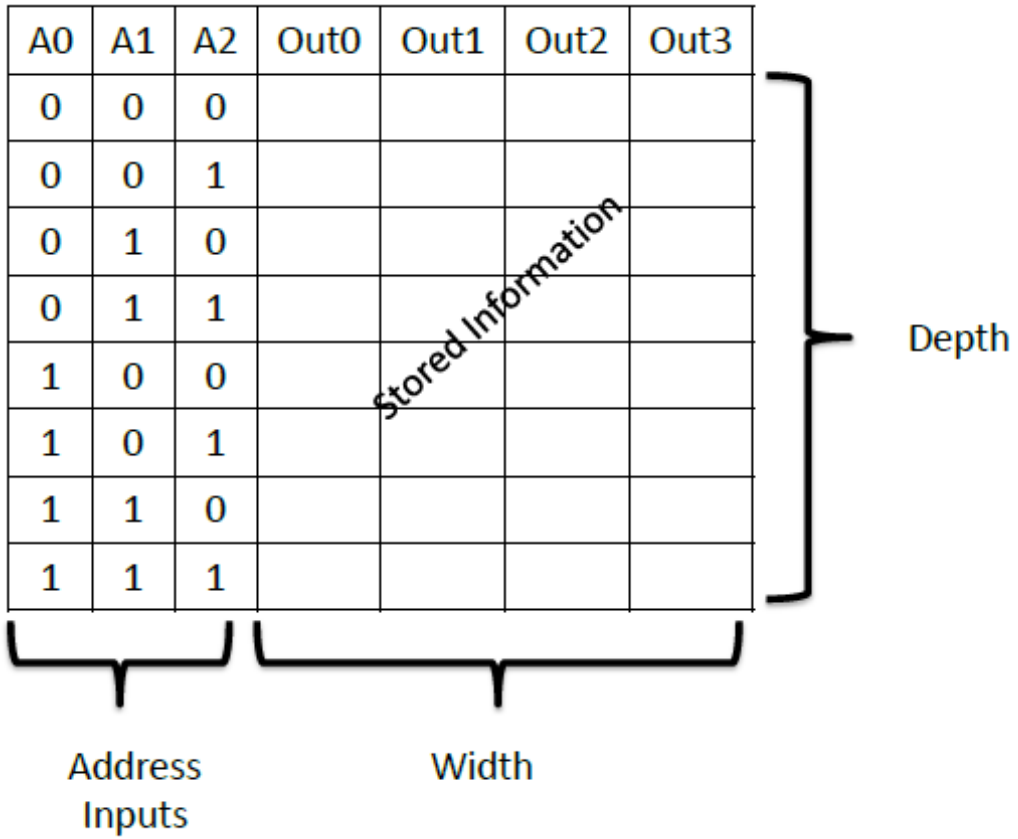
Ingressi

Larghezza (width)



LUT larga 4 e profonda 8

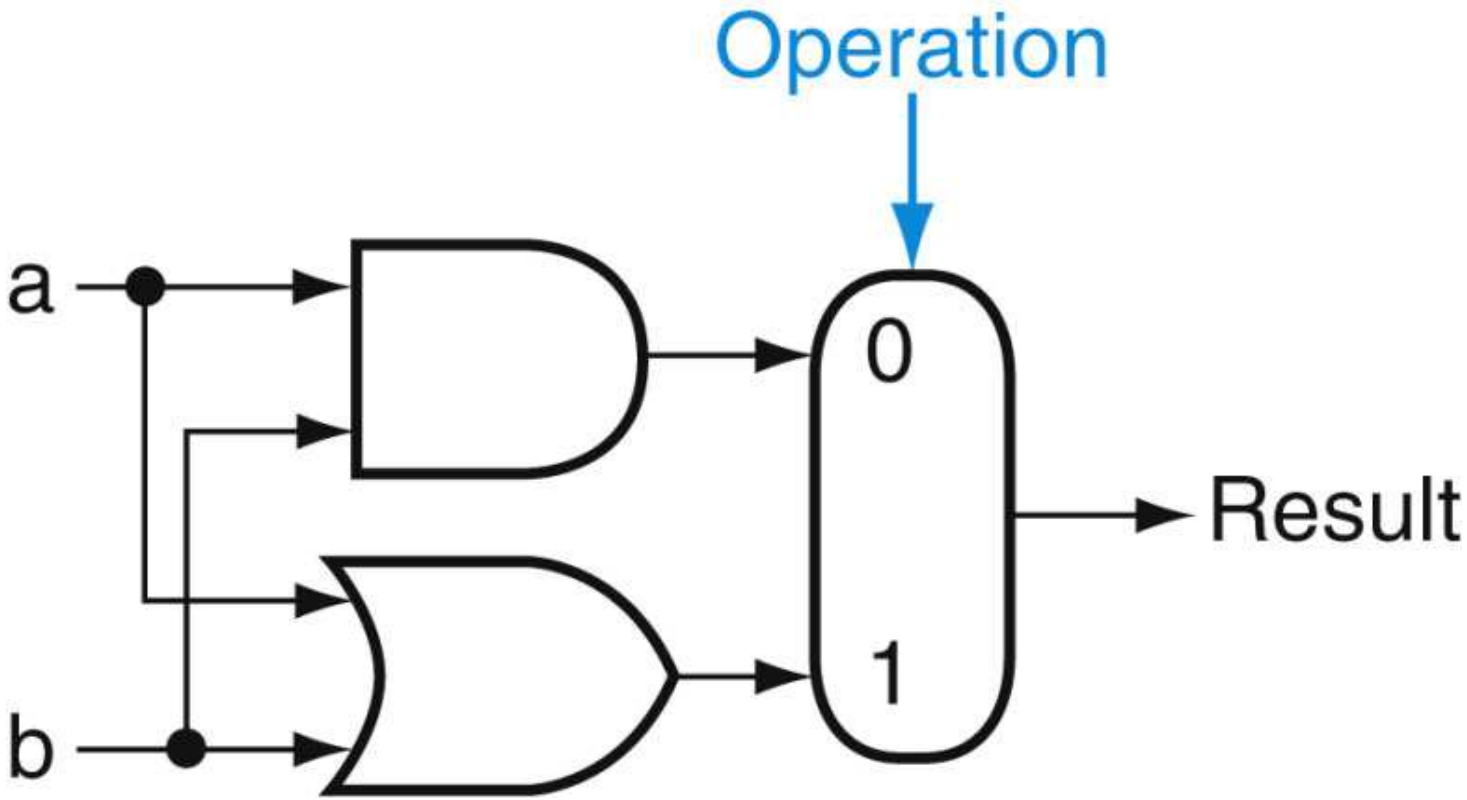
- I segnali di controllo del MUX sono usati per selezionare le possibili uscite della funzione booleana



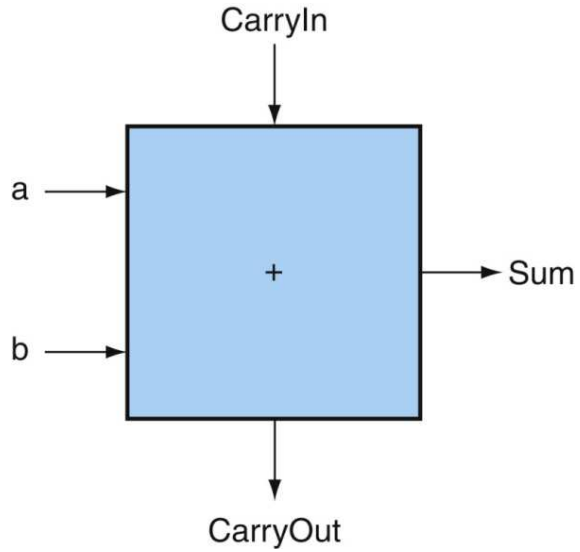
- Vedremo successivamente come rendere tali bit di ingresso "programmabili"
- Le FPGA forniscono un determinato numero di LUTs per realizzare funzioni piu' complesse (anche soft-processors !)

Relizzazione di ALU a 1-bit

- AND e OR



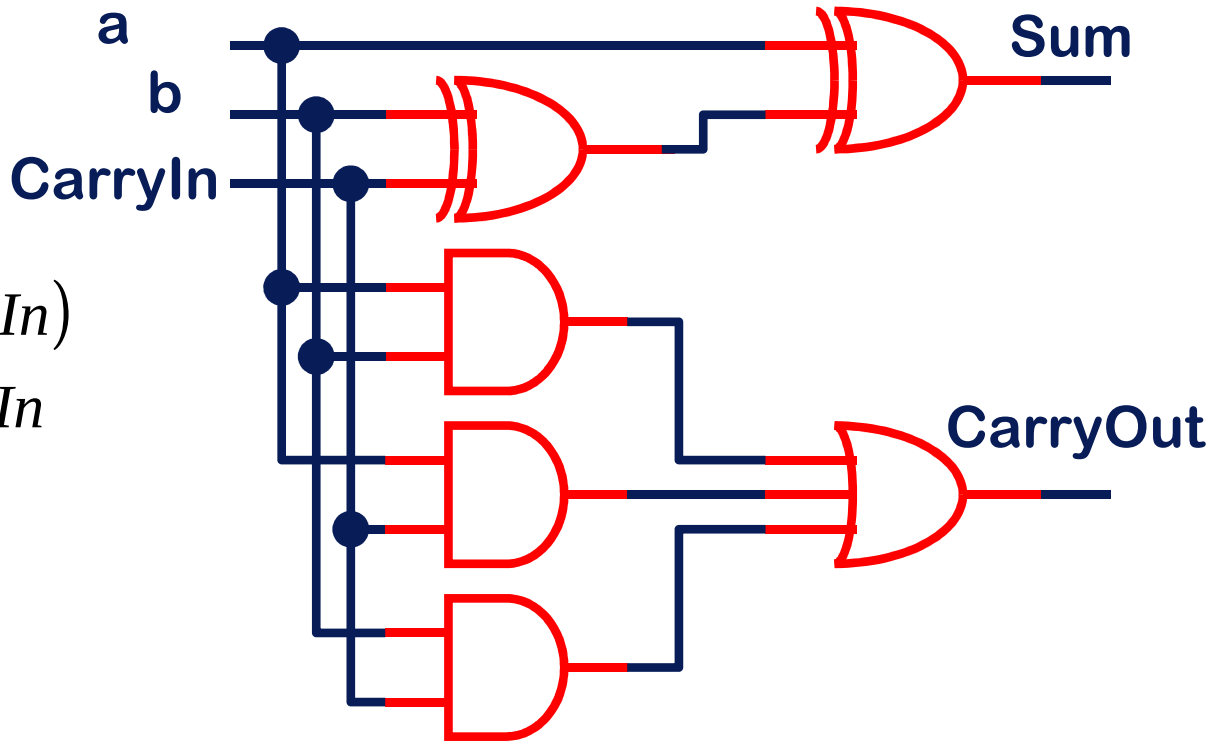
Full Adder a 1 bit



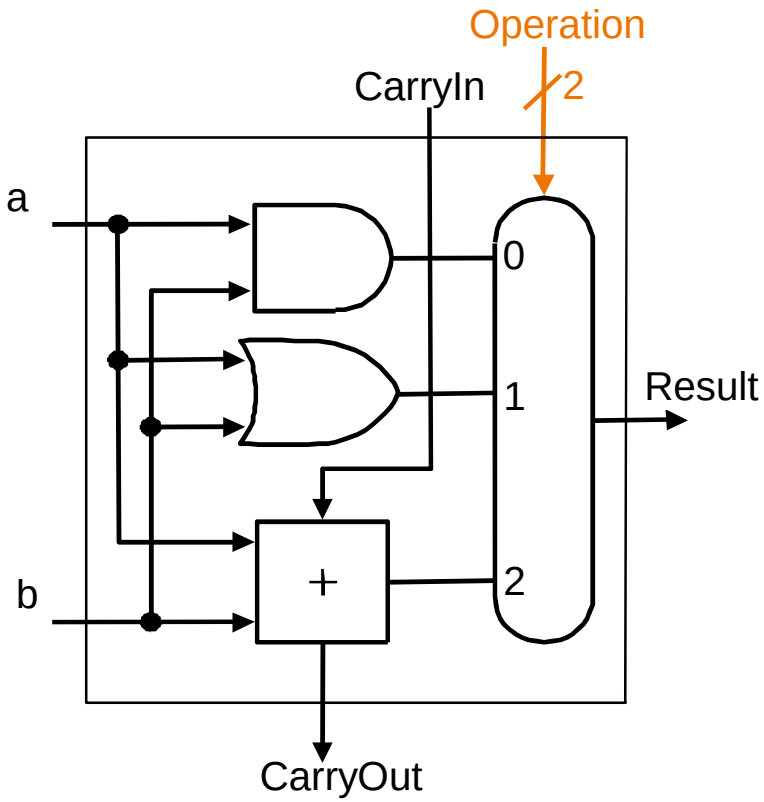
Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

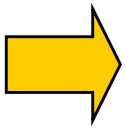
$$sum = a \oplus b \oplus CarryIn = a \oplus (b \oplus CarryIn)$$

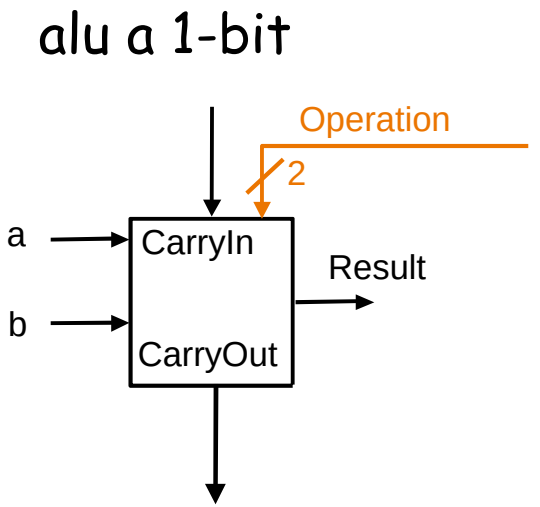
$$CarryOut = a \cdot b + a \cdot CarryIn + b \cdot CarryIn$$



Semplice ALU a 1-bit

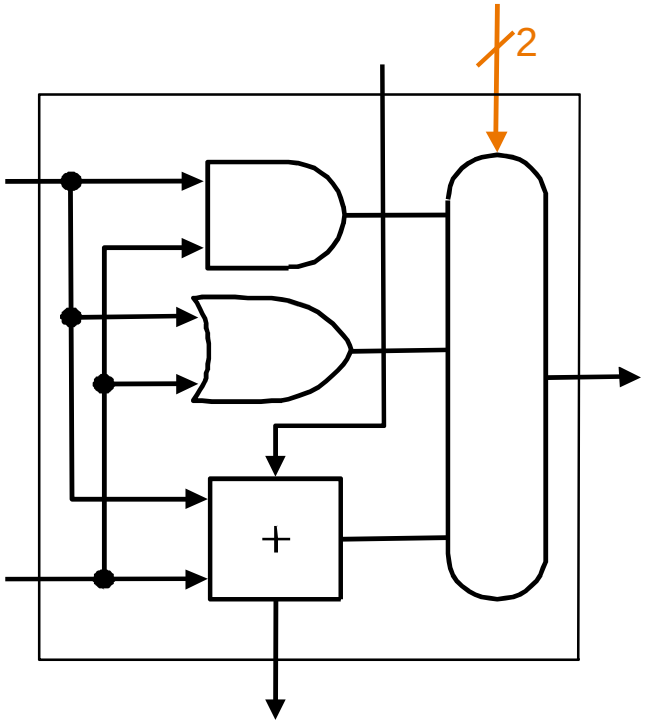


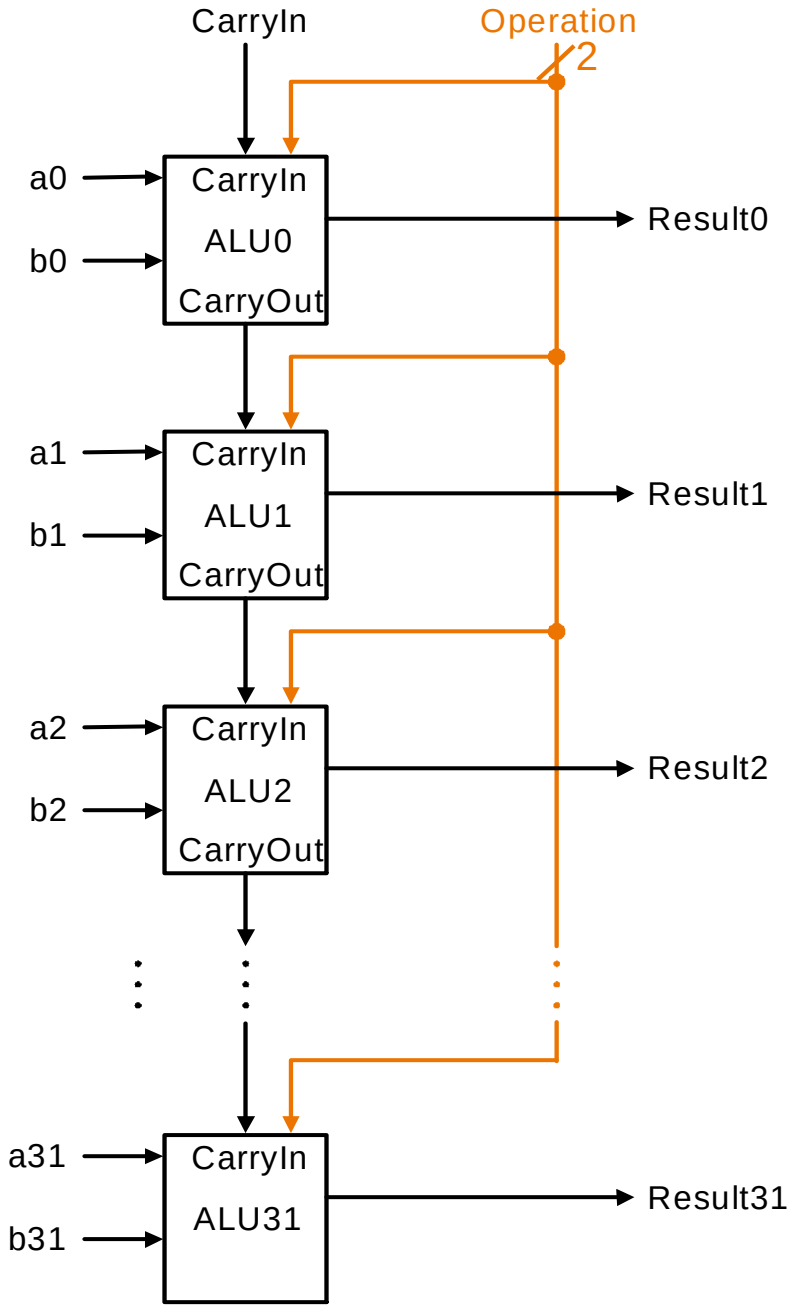




Operation	Function
0	AND
1	OR
2	ADD

Costruire una ALU a 32 bit





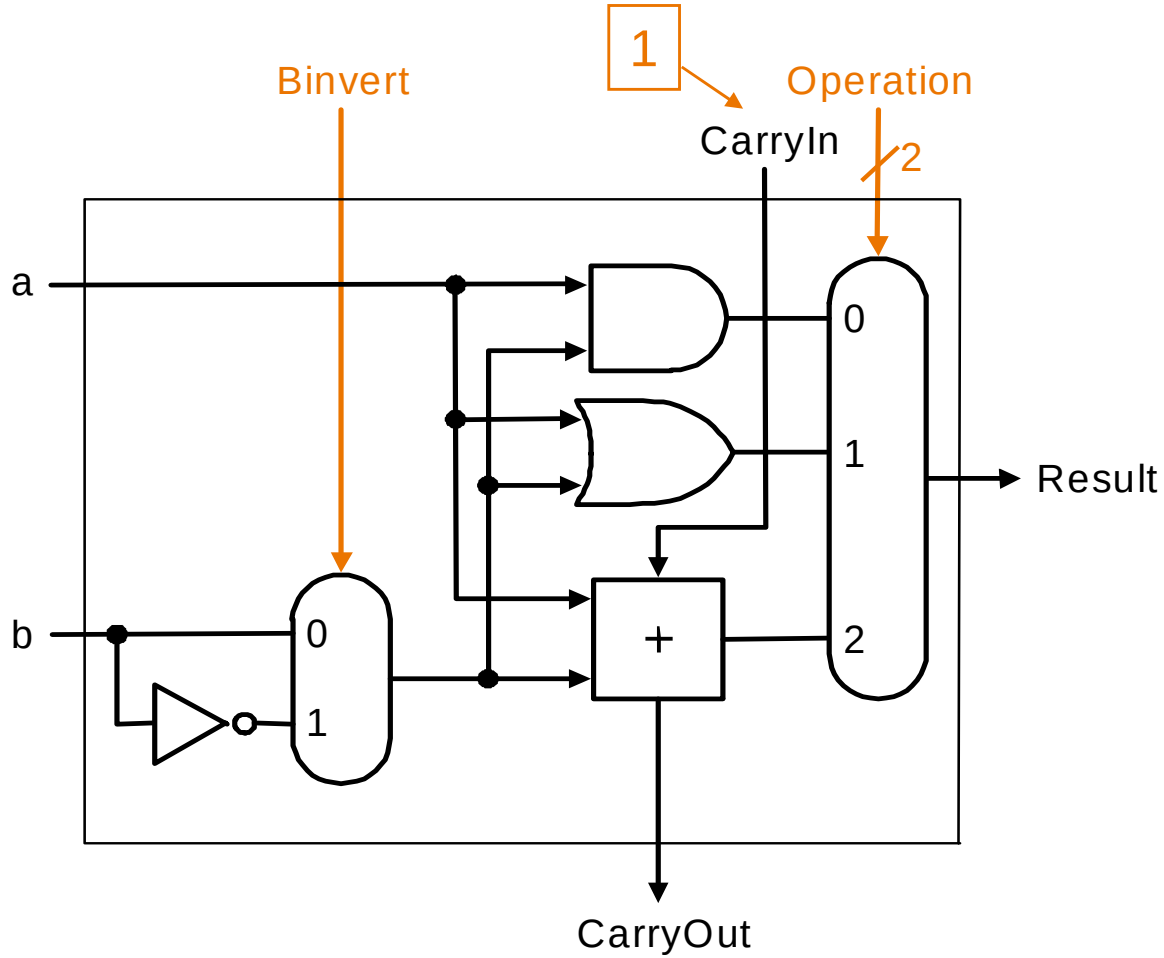
Problema:
il risultato e' disponibile
solo dopo che ALU31
ha ricevuto il carry

Differenti implementazioni del sommatore

- Vedere testo per diverse implementazioni di:
  - Sommatore
  - Sommatore con Look Ahead Carry
- Si puo' risolvere il problema della propagazione del carry utilizzando reti specializzate per la somma

Sottrazione (a – b) ?

- Approccio: fare a + (-b)
- Usare la rappresentazione in complemento a due semplifica la logica necessaria per fare la negazione



Supporto per la `slt` e per il test di uguaglianza

- **Funzione set-on-less-than (`slt`)**
  - **Se a < b produce 1, altrimenti 0**
  - **Idea: usare la sottrazione**
    (a-b) < 0 implica a < b

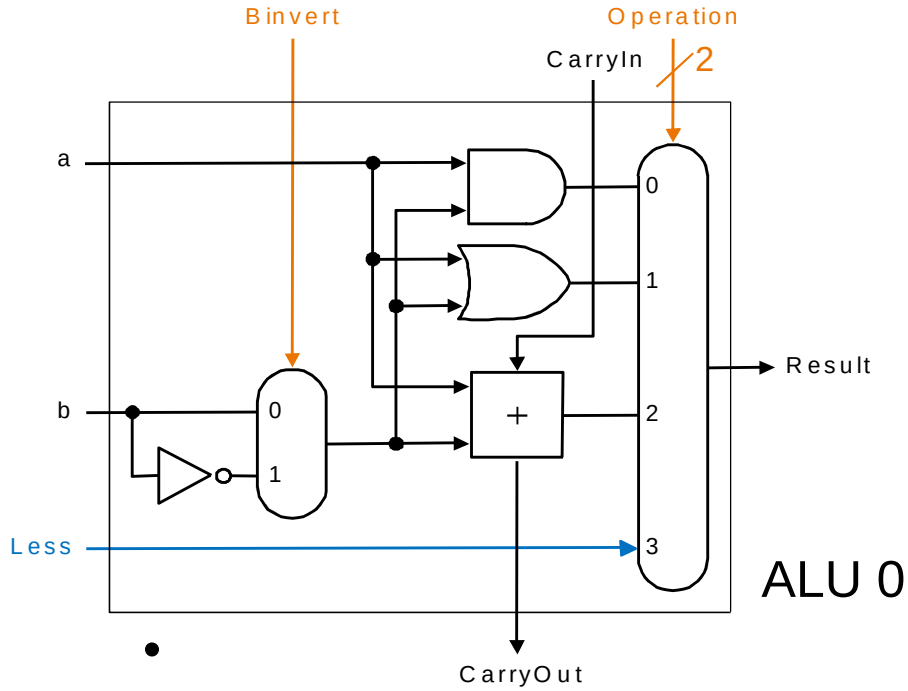
Supporto per il test di uguaglianza

- **Se a == b allora produco 1, altrimenti 0**
- **Idea: usare nuovamente la sottrazione**
  (a - b) = 0 implica a = b

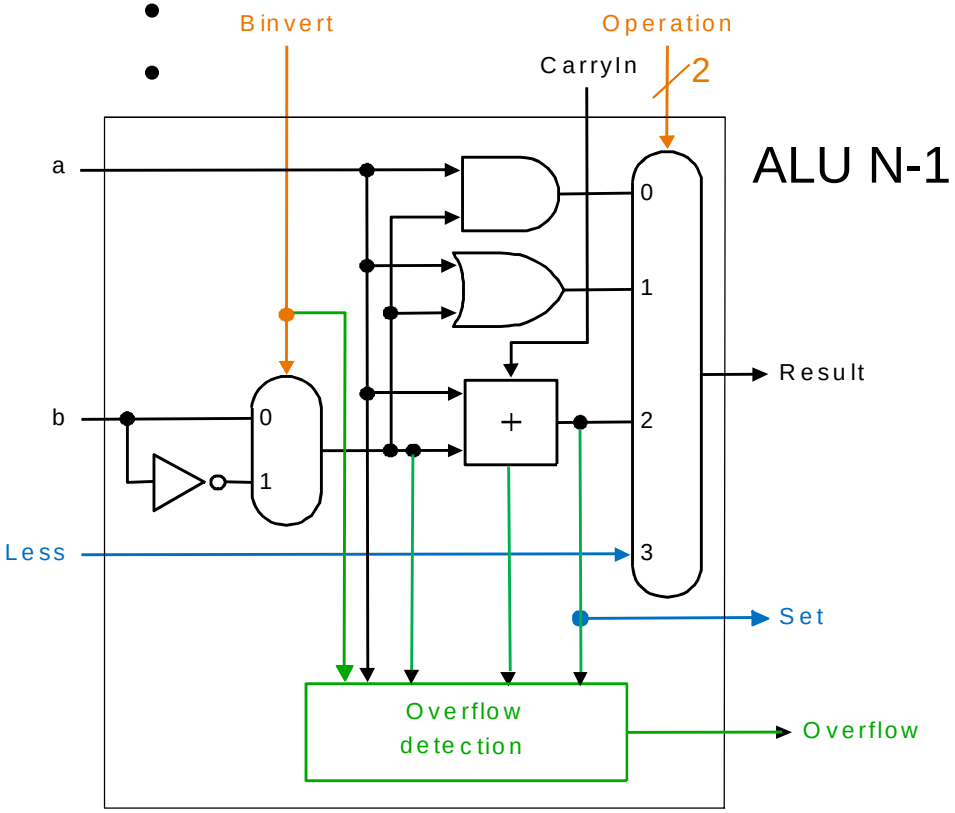
→ In entrambi i casi si puo' si puo' sfruttare la rete per fare la sottrazione

Supporto alla `slt` *nella ALU*

- Inserisco il caso "3" al multiplexer, corrispondente a `slt`
- Nelle ALU 0,1, ... N-1 questo ingresso e' collegato a un filo "Less" che costituisce un "bypass" ingresso-uscita
- Nella ALU N-1 inoltre (bit piu' significativo), controllo il valore di uscita 'set' che indica come e' andato l'esito del confronto a<b
- Mentre per gli interi in complemento a due e' semplice sapere quando il risultato e' negativo (segnale 'set'), in generale per rivelare l'overflow ho bisogno di una rete logica un poco piu' complessa



ALU 0



ALU N-1

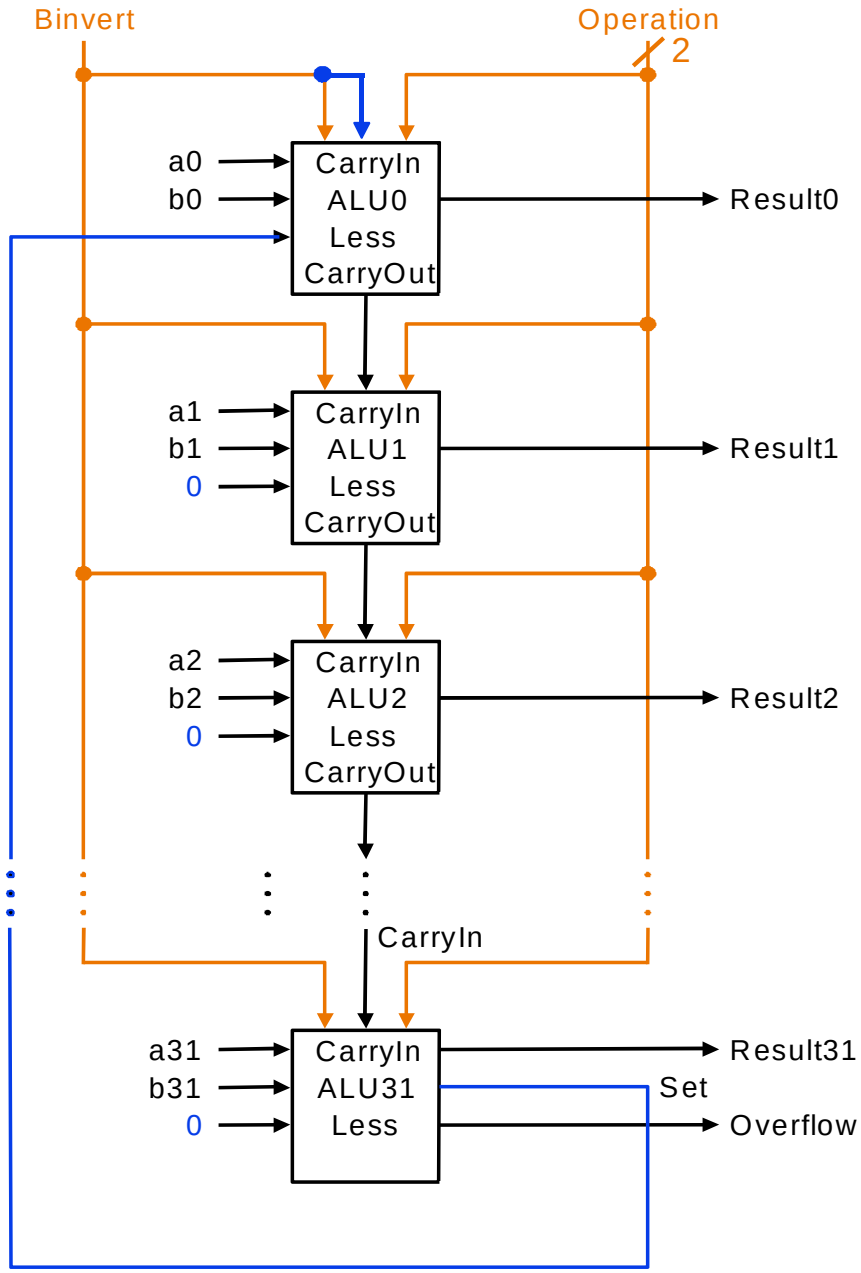
Supporto alla `slt` *fuori* dalla ALU

- Se il confronto ha esito positivo il filo less, manda sul bit meno significativo il risultato (1)

- Aggiustiamo il $CarryIn_0$:

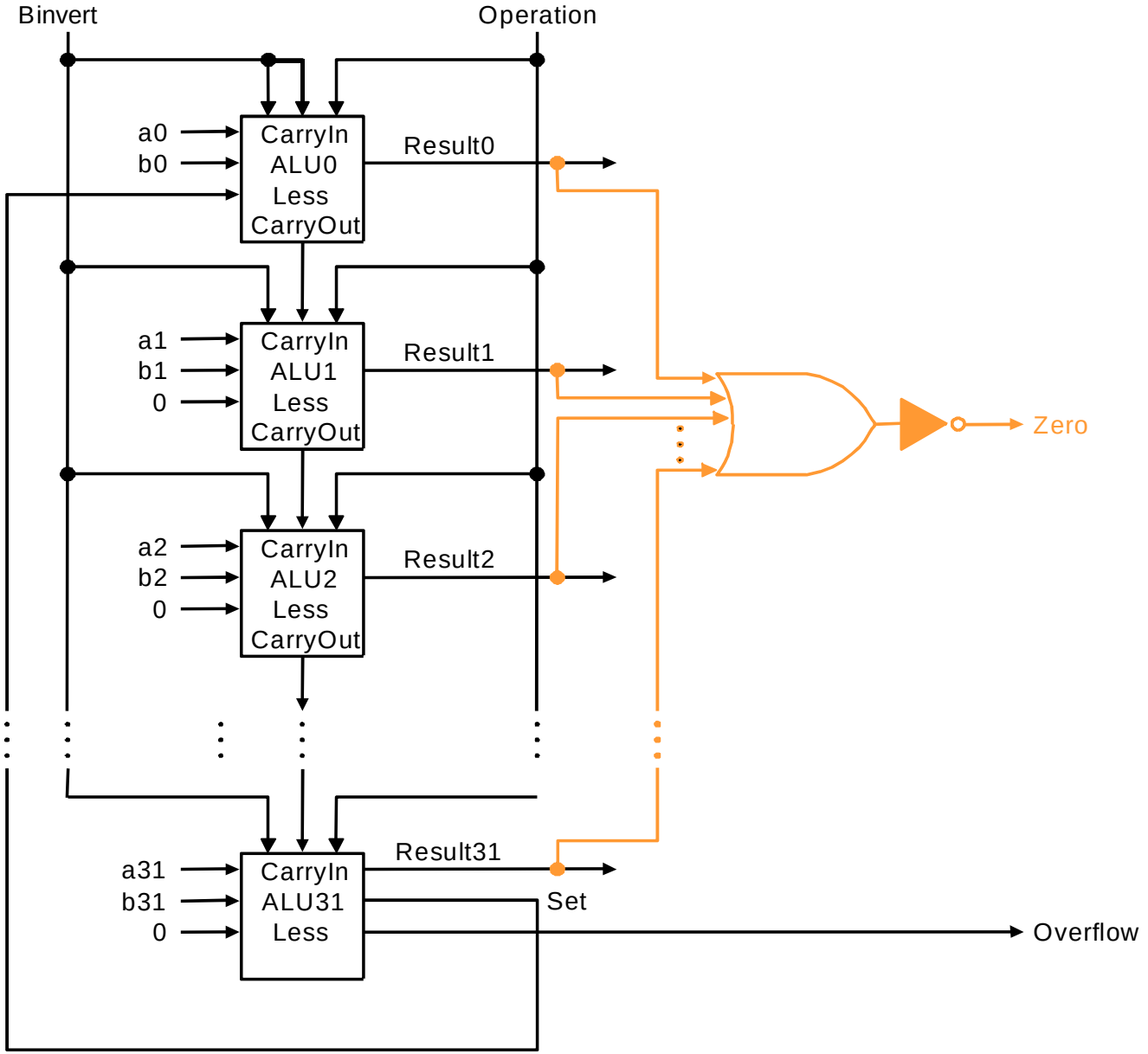
Operation	Function	CarryIn0	BInvert
0	AND	0	0
1	OR	0	0
2	ADD	0	0
2	SUB	1	1
3	SLT	1	1

→ $CarryIn_0$ = BInvert



Test per l'uguaglianza (beq)

- La rete fornisce il valore 1 quando tutti i bit del risultato sono a zero (--> a=b)
- Chiameremo tale segnale "Zero" e ci serve per decidere se l'esito del confronto e' positivo o meno

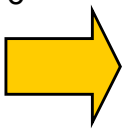


•*Nota: "Zero" vale 1 quando il risultato e' zero!*

ALUcontrol

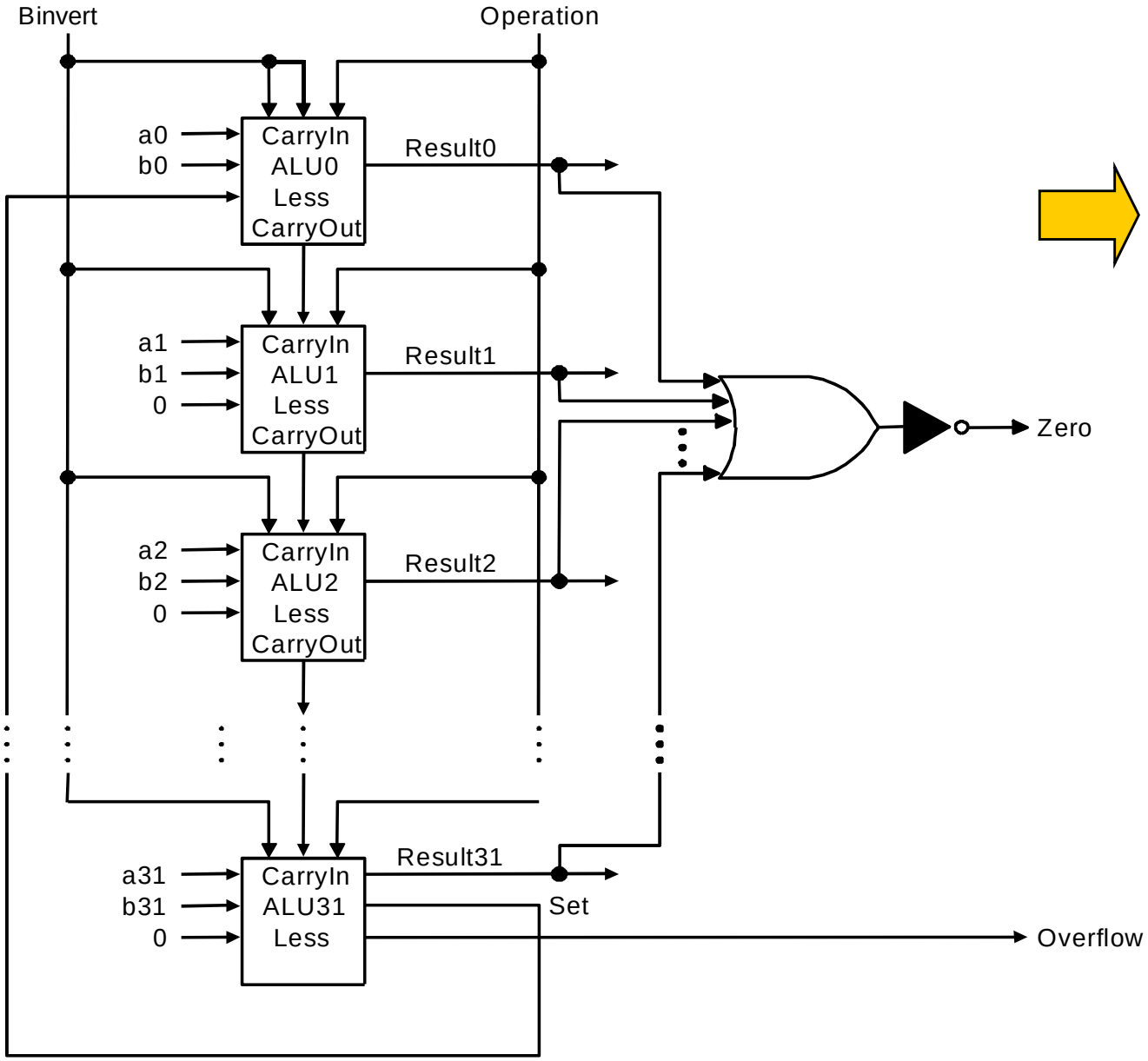
Operation	Function	BInvert
0	AND	0
1	OR	0
2	ADD	0
2	SUB	1
3	SLT	1
2	EQ	1

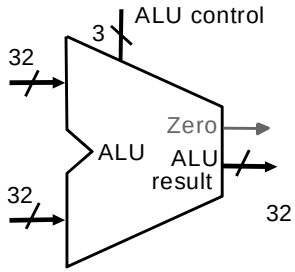
Binvert → $ALUcontrol_2$
$Operation_1$ → $ALUcontrol_1$
$Operation_0$ → $ALUcontrol_0$



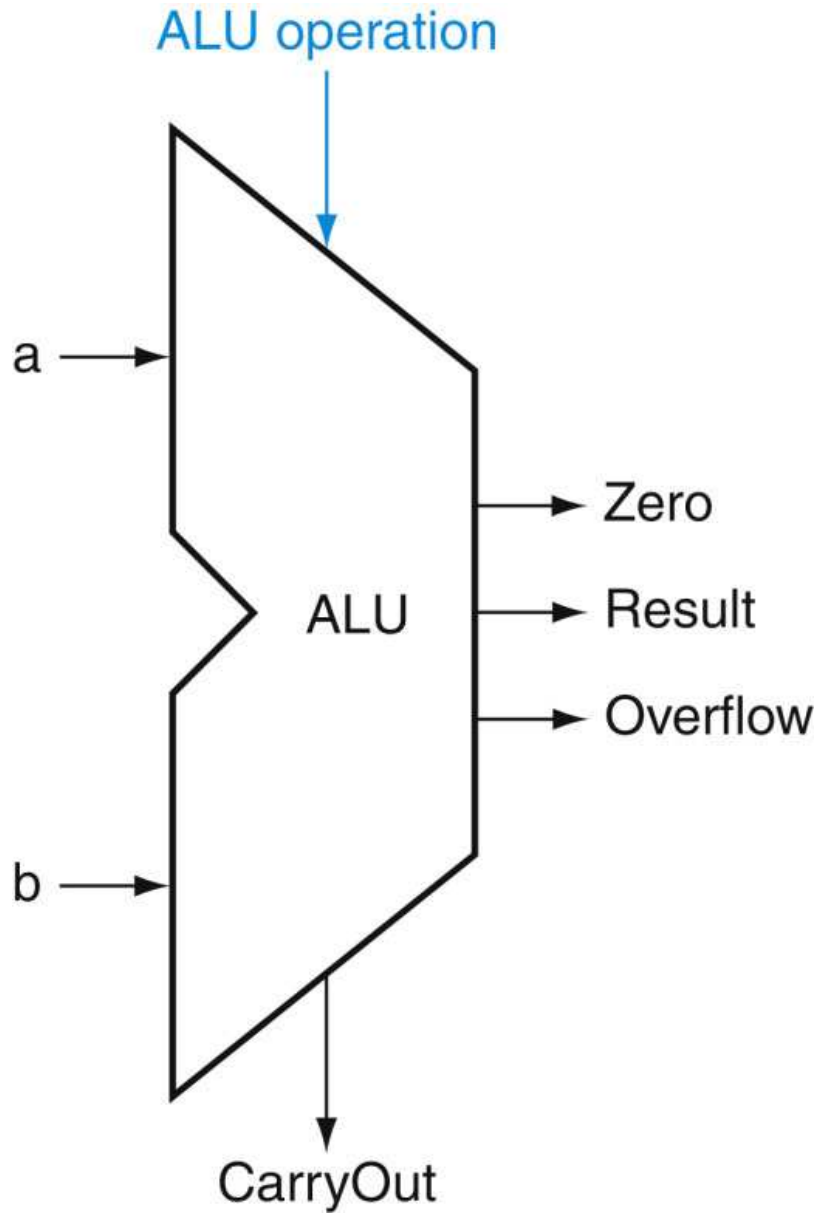
ALUcontrol	
0	00
0	01
0	10
1	10
1	11
1	10

Blocco "ALU a 32-bit"





Overflow lo possiamo dare per implicito nel blocco ALU


ALU operation
a
b
ALU
Zero
Result
Overflow
CarryOut