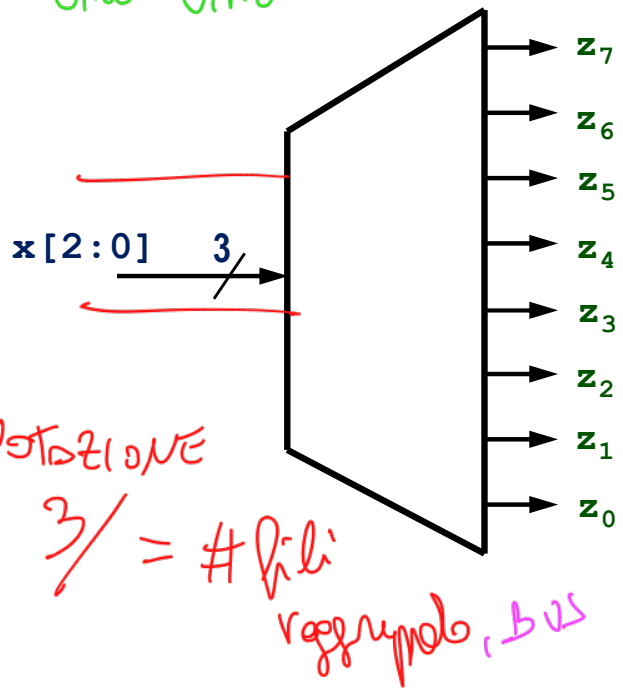
Lezione 3
Elementi Architetturali Principali: Reti Combinatorie Notevoli

(v. PHRV1, appendice A.2,A.3,A.5,A.12)

- Decoder
- Encoder
- Encoder con priorità
- Multiplexer
- Demultiplexer
- Look-Up-Table (LUT)
- Arithmetic-Logic Unit (ALU)

Decoder

(v. PHRV1, appendice A.3)



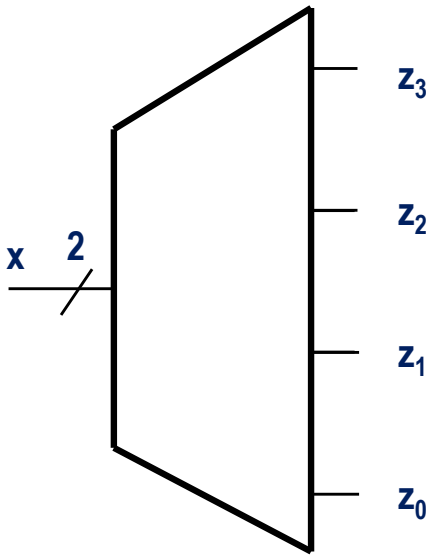
INGRESSI			USCITE							
x_2	x_1	x_0	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

- **Esempio di decoder a 3 bit o "da-3-a-8"**
- **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

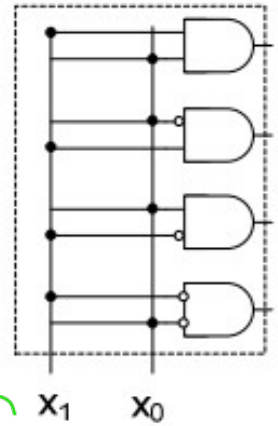
2 ingressi
da 2 a 4

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



Es. decoder da-2-a-4

tutte e 4 varianti di porte and



z_3 uscita = 1
z_2 $x_1 = 0$ $x_0 = 1$
z_1 $x_1 = 1$ $x_0 = 0$
z_0 0 0

Porta AND in cui vado a negare gli ingressi quando intendo sia 0, e li prendo non negati quando intendo mandi a 1

Implementato con porte logiche, ma non è l'unica implementazione. CMOS è più efficiente

ha invertito x_1 e x_0

decoder da 2 a 4

x_0	x_1	z_3	z_2	z_1	z_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Nota: in realtà possono esistere implementazioni fisiche CMOS diverse è più efficienti dell'elemento architetturale "decoder"

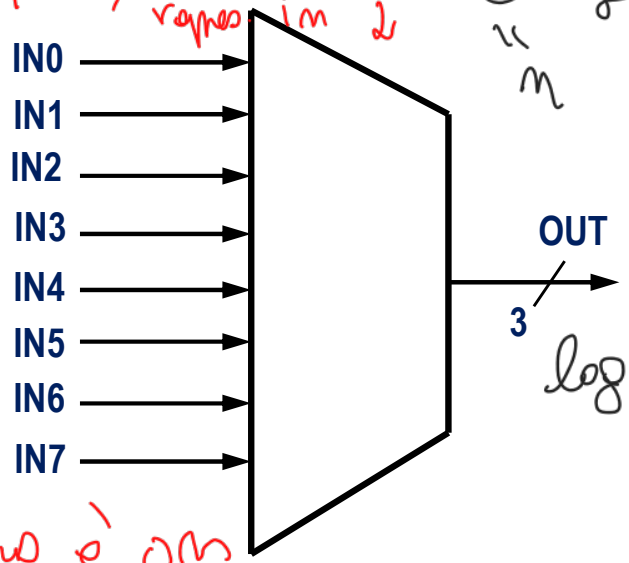
Filo in AND a tutto il resto. uscita abilitata solo se abilitaz. = 1

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

tutto con AND in ingresso, e uscita vale solo se AND è 1 (?)

Encoder

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



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 è una tabella della verità; la si può interpretare come tabella di verità osservando che nelle mancanti 244 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 può essere realizzato con 3 porte OR a 4 ingressi

Esercizio per casa: è possibile ricavare queste equazioni dalle mappe di Karnaugh ?

Esercizio: Encoder da 4 a 2

- 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

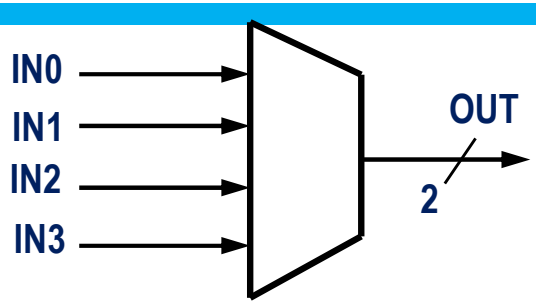


TABELLA DI VERITÀ (COMPLETA):

IN3	IN2	IN1	IN0		OUT1	OUT0
0	0	0	1		0	0
0	0	1	0		0	1
0	1	0	0		1	0
1	0	0	0		1	1
0	0	0	0		X	X
0	0	1	1		X	X
0	1	0	1		X	X
0	1	1	0		X	X
0	1	1	1		X	X
1	0	0	1		X	X
1	0	1	0		X	X
1	0	1	1		X	X
1	1	0	0		X	X
1	1	0	1		X	X
1	1	1	0		X	X
1	1	1	1		X	X

IN3,IN2 \ IN1,IN0	00	01	11	10
00	X	0	X	1
01	0	X	X	X
11	X	X	X	X
10	1	X	X	X

OUT0

IN3,IN2 \ IN1,IN0	00	01	11	10
00	X	0	X	0
01	1	X	X	X
11	X	X	X	X
10	1	X	X	X

OUT1

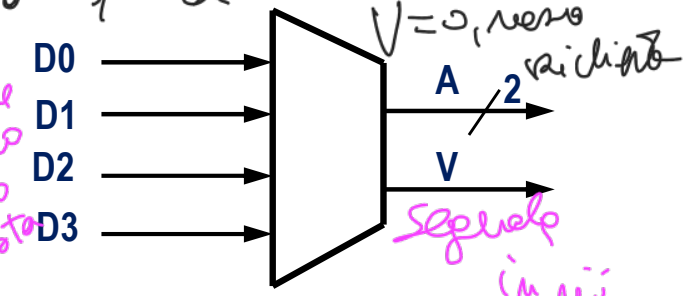
OUT0=IN1+IN3 OUT1=IN2+IN3

Priority Encoder

- **Questa è 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

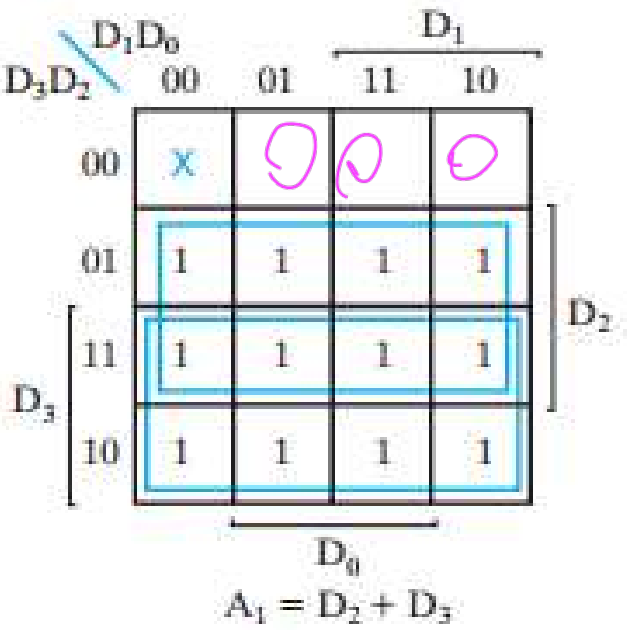
Tabella di verità completa

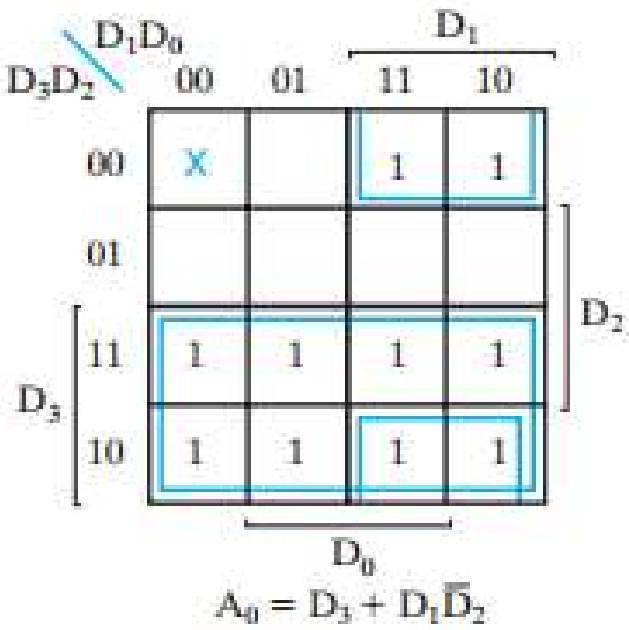


- **Quando D3 vale 1 tutti gli altri bit non contano e l'uscita indica la massima priorità $(11)_2$ ovvero 3**
- **Quando D3 vale 0 ma D2 vale 1 gli altri bit a destra non contano e l'uscita indica priorità 2 e così via…**
- **Il bit V è necessario per indicare se l'uscita è valida ovvero se almeno un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

Nota: stavolta gli 'X' (non-specificato) sugli INPUTS assumono il significato di "sia nel caso 0 che nel caso 1".

Realizzazione del Priority Encoder



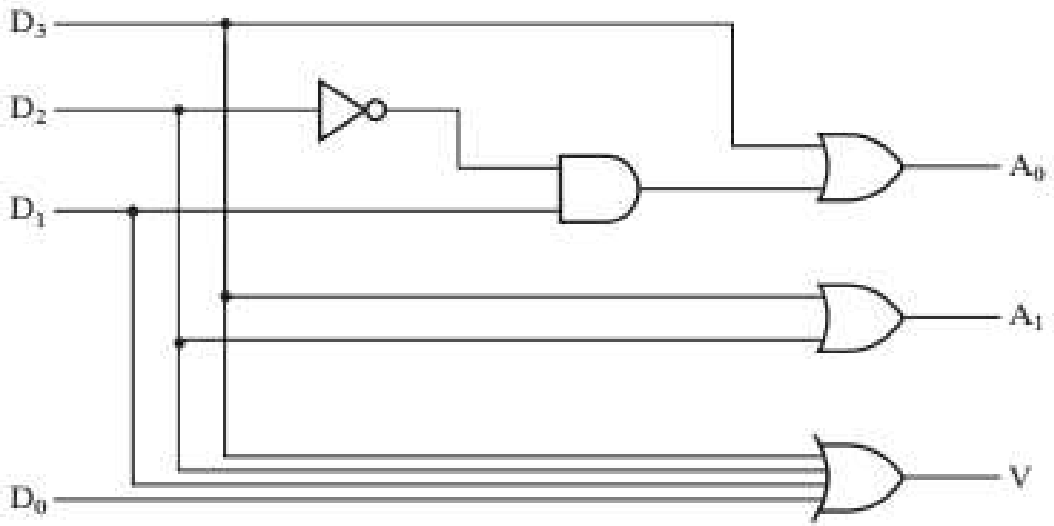


$A_0 = D_3 + D_1\overline{D}_2$

$A_1 = D_2 + D_3$

$V = D_0 + D_1 + D_2 + D_3$

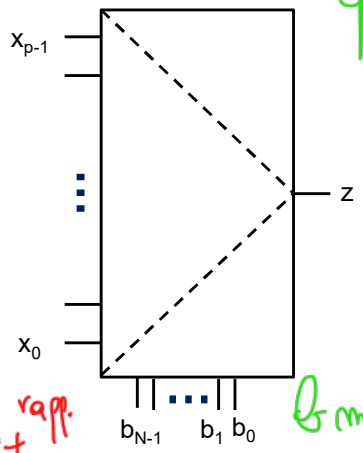
Con MAXTERM



Uscita vale X se ...
... la rappres. binaria di B vale ...

Multiplexer

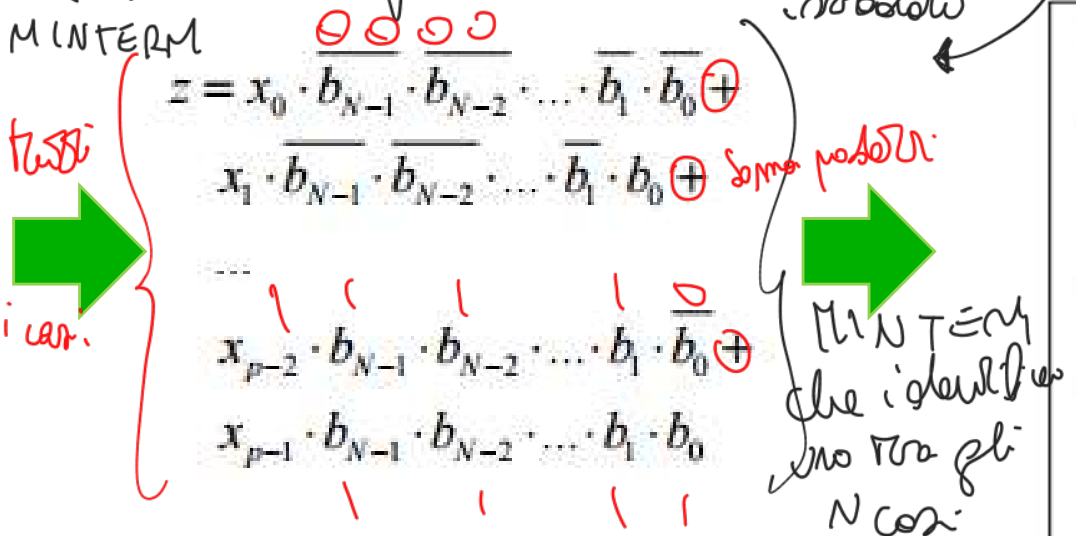
- **Il multiplexer serve per selezionare un'informazione**

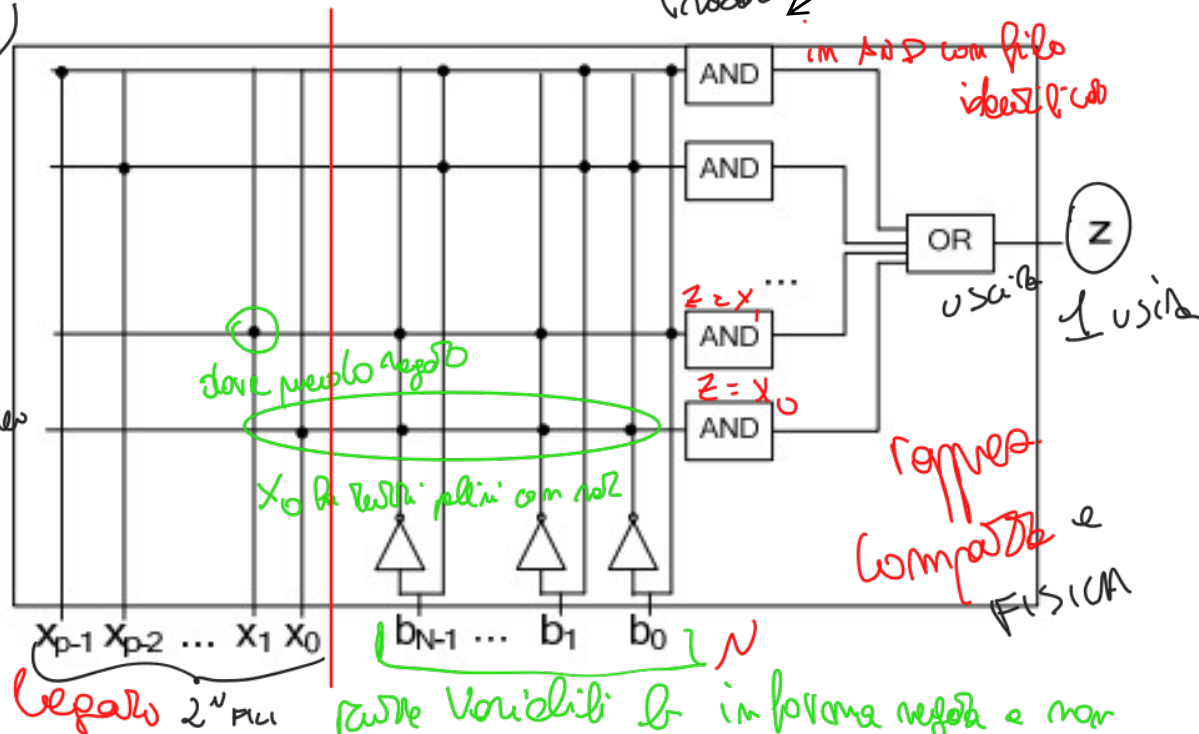


Multiplexer con 2^N ingressi

- **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**

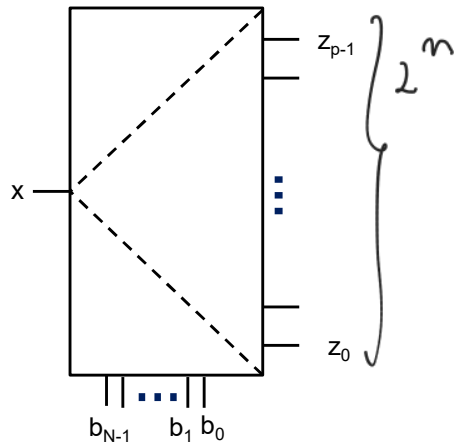
$$z = x_i \Leftrightarrow (b_{N-1} \ldots b_1 b_0)_2 \equiv i$$





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

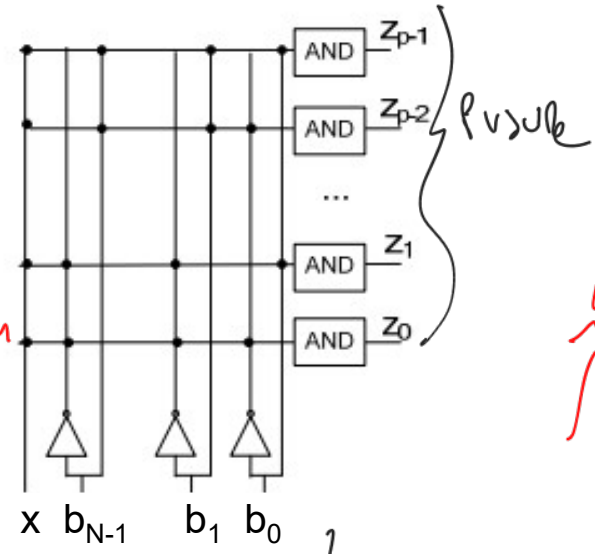
$$z_0 = x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \ldots \cdot \overline{b_1} \cdot \overline{b_0}$$

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

...

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

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

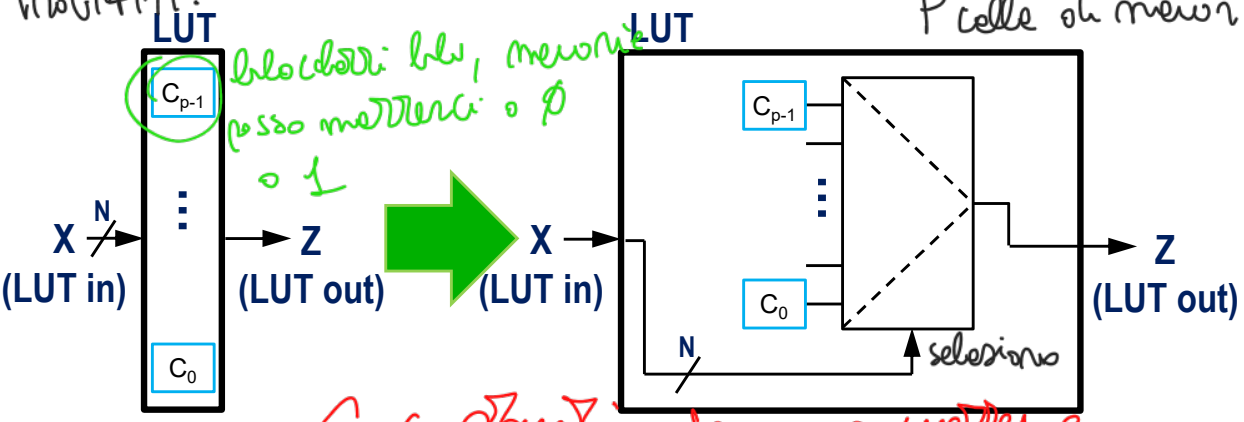


Può essere visto come un decoder con abilitazione

Look Up Table (LUT)

(v. PHRV1, appendice A.12)

- è un Multiplexer in cui gli ingressi sono costanti



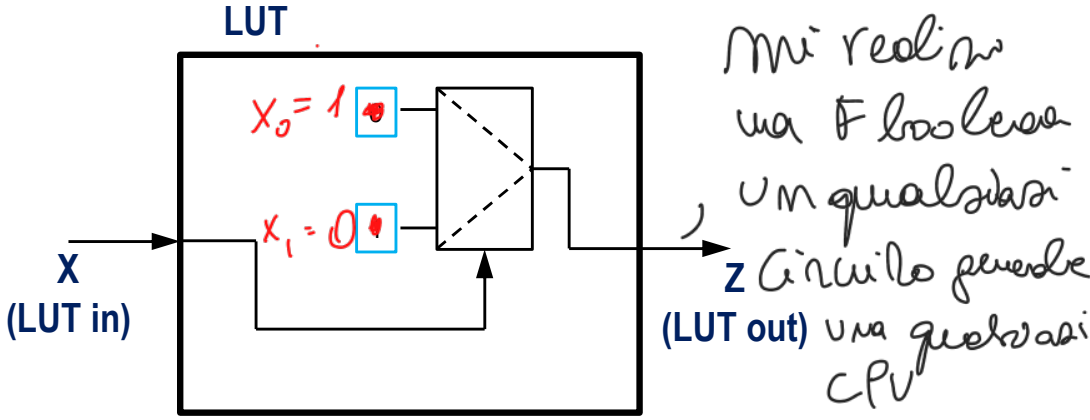
- Il multiplexer ha (con $p=2^N$) 2^N+N ingressi e 1 uscita
- Gli N ingressi X fungono da variabili di comando
- Specificando X=i viene selezionato uno (quello con indice i) dei $2^N=p$ ingressi (FISSI ovvero COSTANTI)
- è utile per realizzare una funzione booleana generica
- Esempio realizzazione della funzione "inverter" tramite LUT

X	Z
0	1
1	0

Profondità (depth)

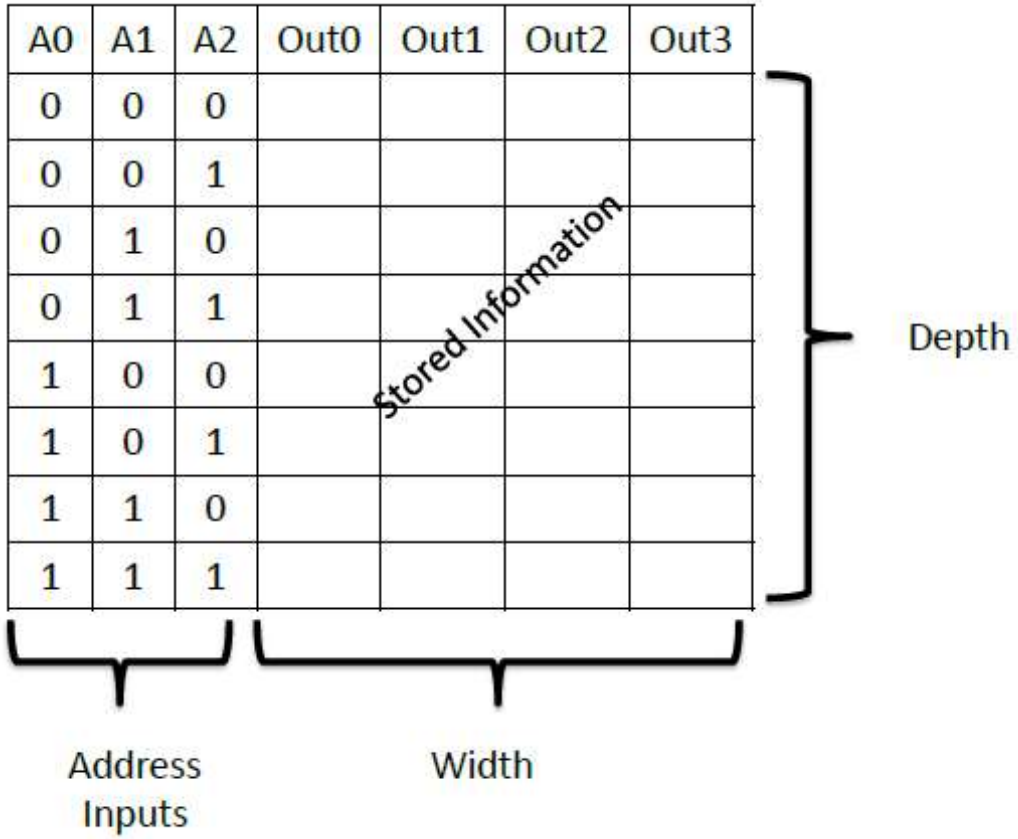
Ingressi

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" (usando» celle di memoria»)
- Le FPGA forniscono un determinato numero di LUTs per realizzare funzioni più complesse (anche soft-processors !)

COSTRUZIONE DI UNA ARITHMETIC LOGIC UNIT (ALU)

(V. PHRV1, APPENDICE A.5)

Cuore del Processore

ALU cuore di un processore, ha le 4 operazioni (somma, sott, AND e OR)

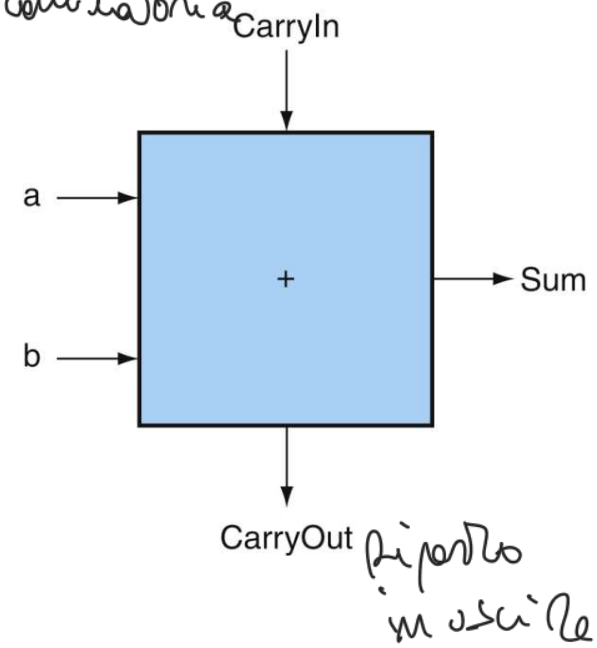
Con riporto

Full Adder a 1 bit

circuito che fa la somma di 2 bit a e b

(v. PHRV1, appendice A.5)

rete combinatoria

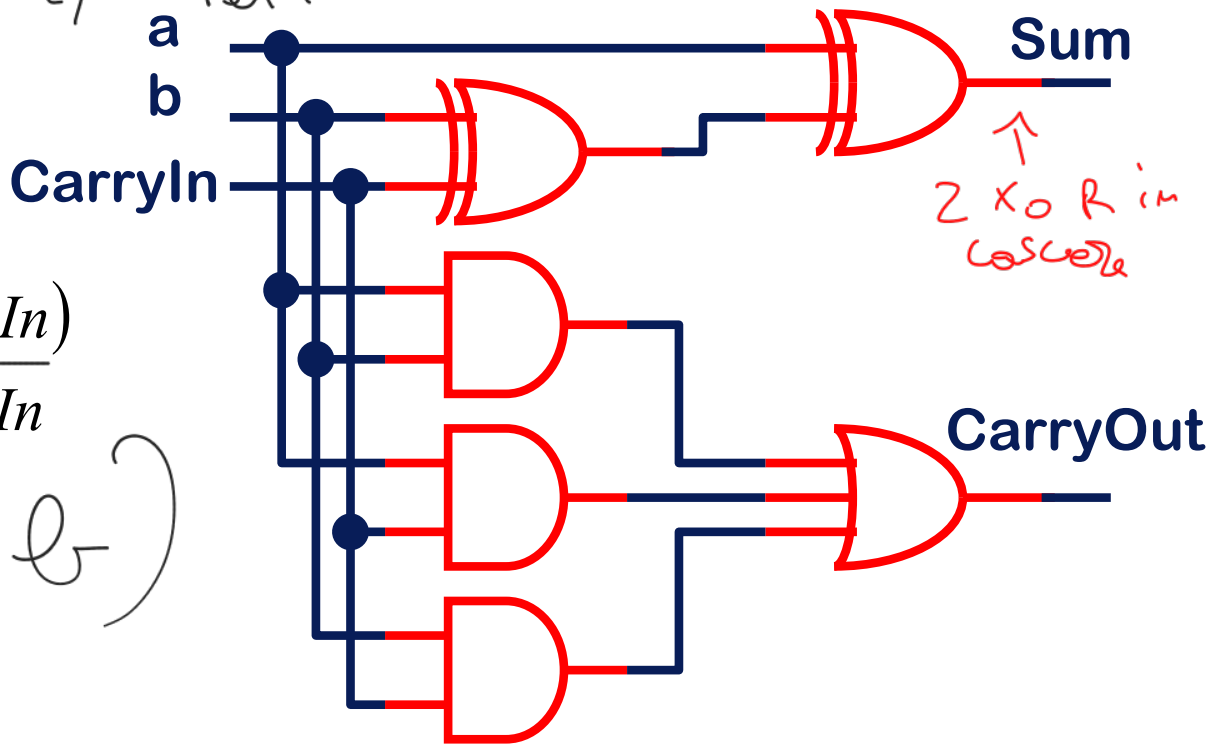


riporto in uscita

riporto in ingresso

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

2 uscite, 2 reti



2 XOR in cascata

SOMMA = XOR INGRESSI

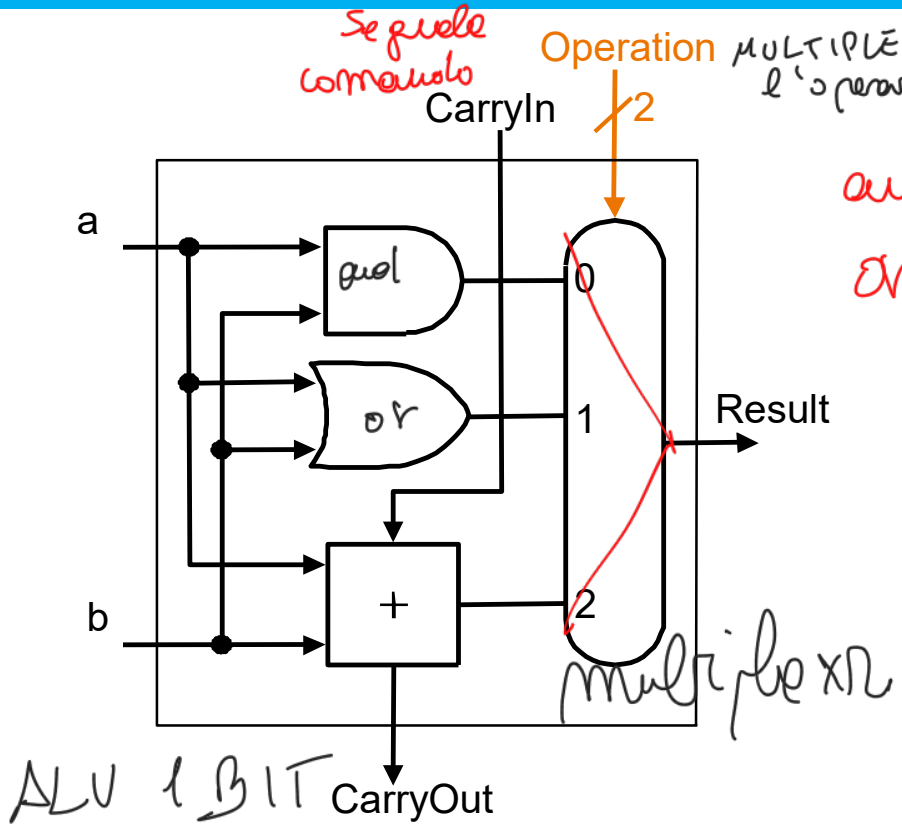
$$sum = a \oplus b \oplus CarryIn = a \oplus (b \oplus CarryIn)$$

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

$a \cdot b + C(a \oplus b)$

Semplice ALU a 1-bit

AND = operazione 0 OR = op. N.1 ADD = N.2



Segnale comando

MULTIPLEXER per selezionare l'operazione

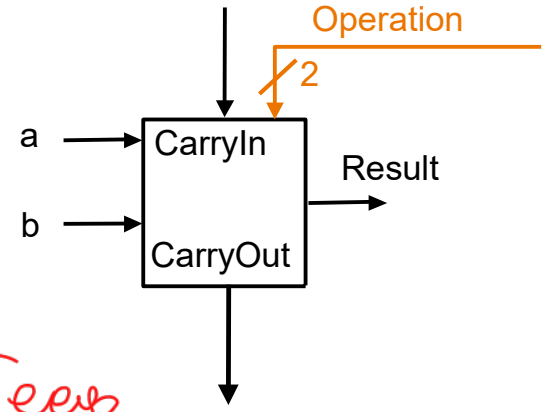
operazione

and a 0

or a 1

Multiplexer

ALU 1 BIT

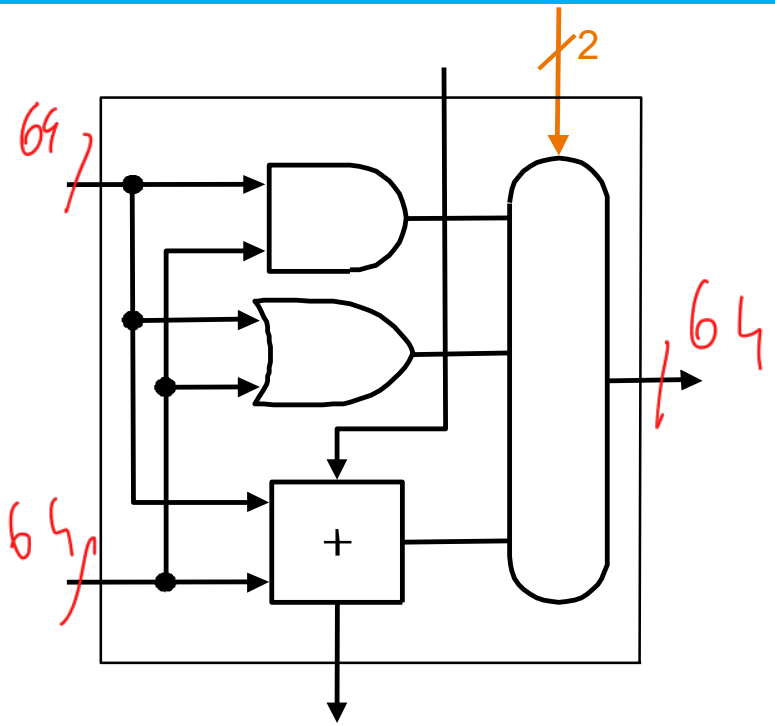


Full Adder = Tiene conto del riporto

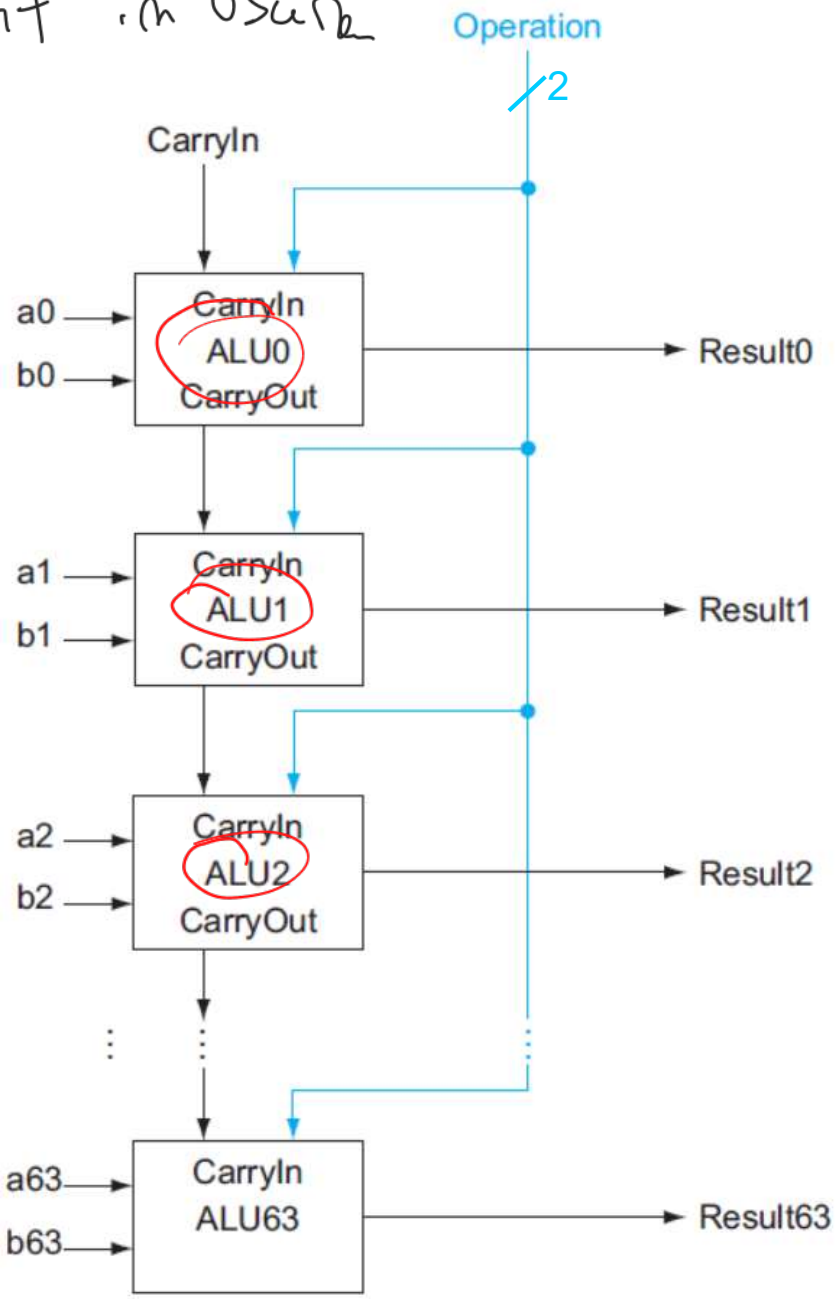
Operation	Function
0	AND
1	OR
2	ADD

operation = segnale di comando, mi seleziona l'operazione

Costruire una ALU a 64 bit



da 1 a 64 bit



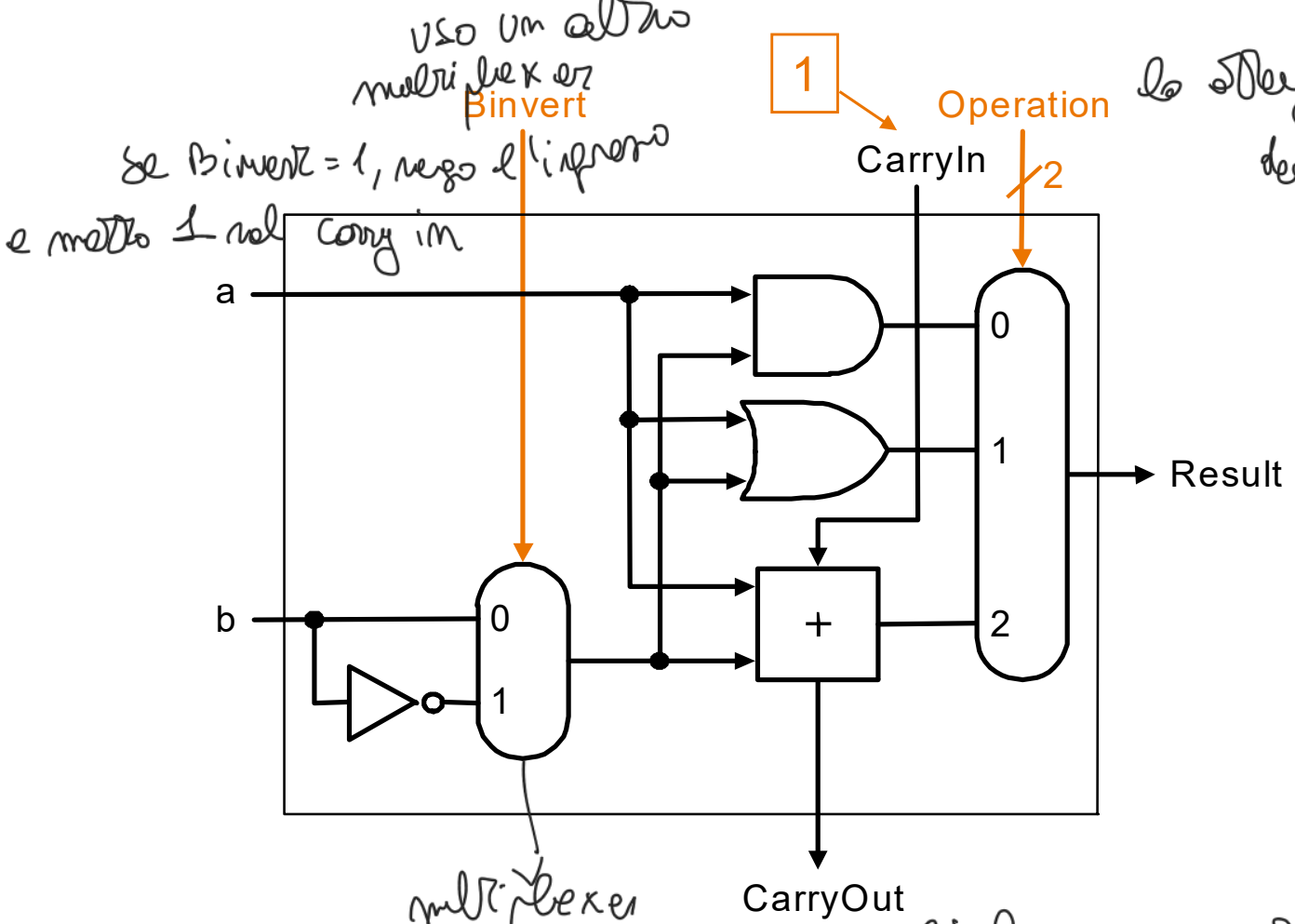
Un sommatore costruito in questo modo è chiamato "sommatore con riporto seriale"

Problema:
il risultato è disponibile solo dopo che ALU63 ha ricevuto il carry

Altre implementazioni "con riporto parallelo" o "Carry Look Ahead" superano questo problema (v. Lezione-06 e PHRV1 appendice A.6)

Supporto per la sottrazione (a - b)

- **Approccio: fare a + (-b)**
- **Usare la rappresentazione in complemento a due semplifica la logica necessaria per fare la negazione: -b = ~b+1**



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
  - Per vedere se (a-b) è negativo basta che verifico se il bit più significativo vale 1

Supporto per il test di uguaglianza

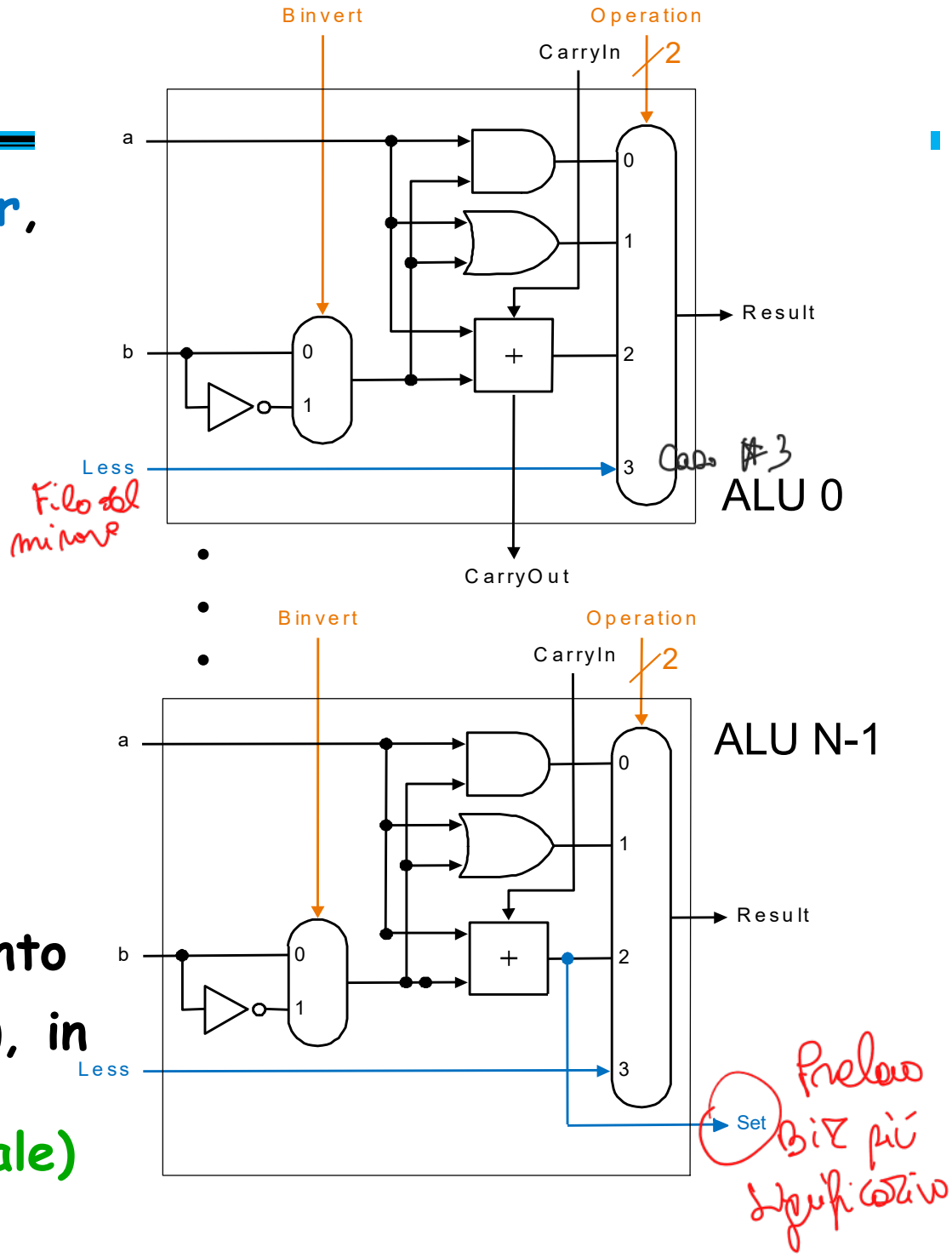
- Ci servirà per supportare le condizioni degli statement if/while/for
  per decidere se devo eseguire quanto segue o «saltare» a un altro pezzo di codice
- Se a == b allora produco 1, altrimenti 0
- Idea: usare nuovamente la sottrazione

  (a - b) = 0 implica a = b

→ In entrambi i casi si può si può 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 è collegato a un filo "Less" che costituisce un "bypass" ingresso-uscita
- Nella ALU N-1 inoltre (bit più significativo), controllo il valore di uscita 'set' che indica come è andato l'esito del confronto a<b
- Mentre per gli interi in complemento a due è semplice sapere quando il risultato è negativo (segnale 'set'), in generale per rivelare l'overflow ho bisogno di una rete logica un poco più complessa (v. slide finale)



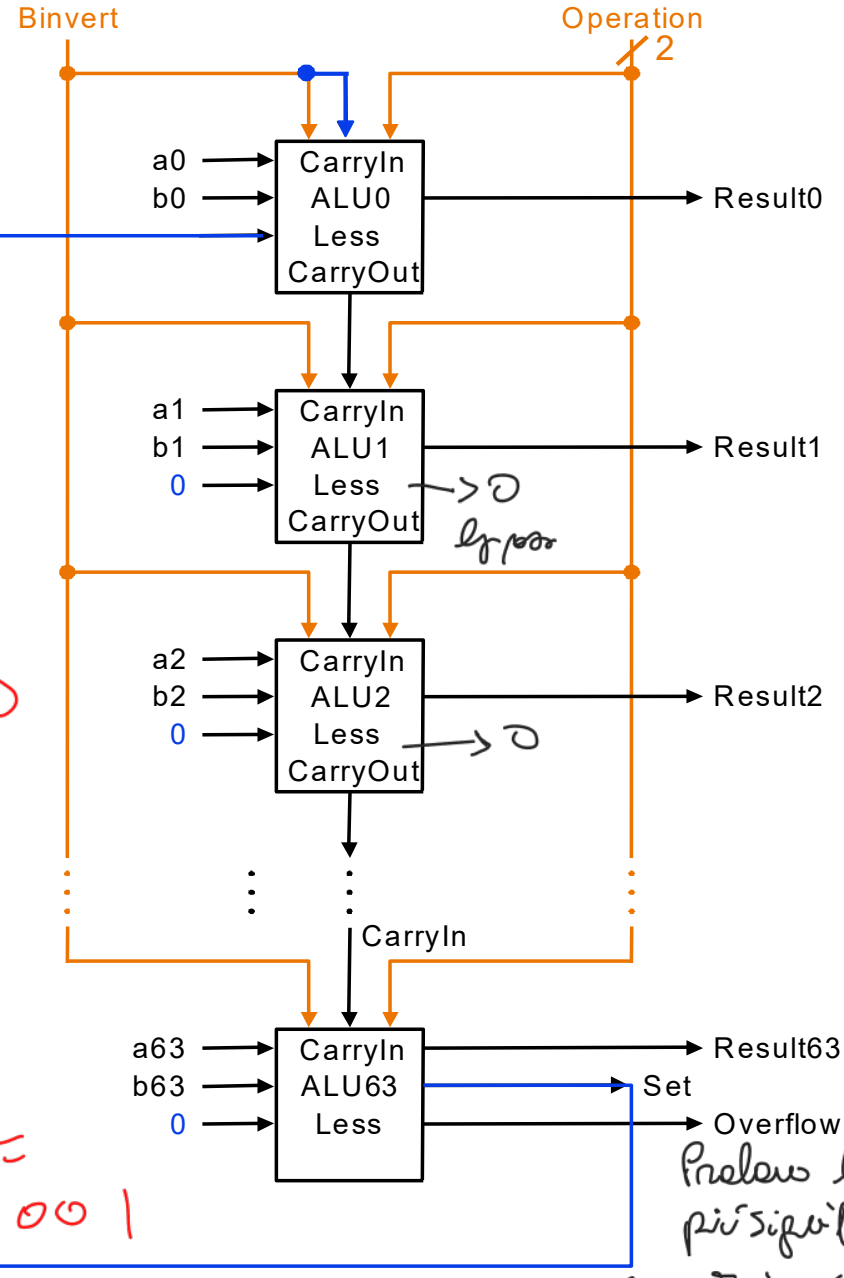
Supporto alla `slt` *fuori* dalla ALU

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

- Aggiustiamo il $CarryIn_0$:

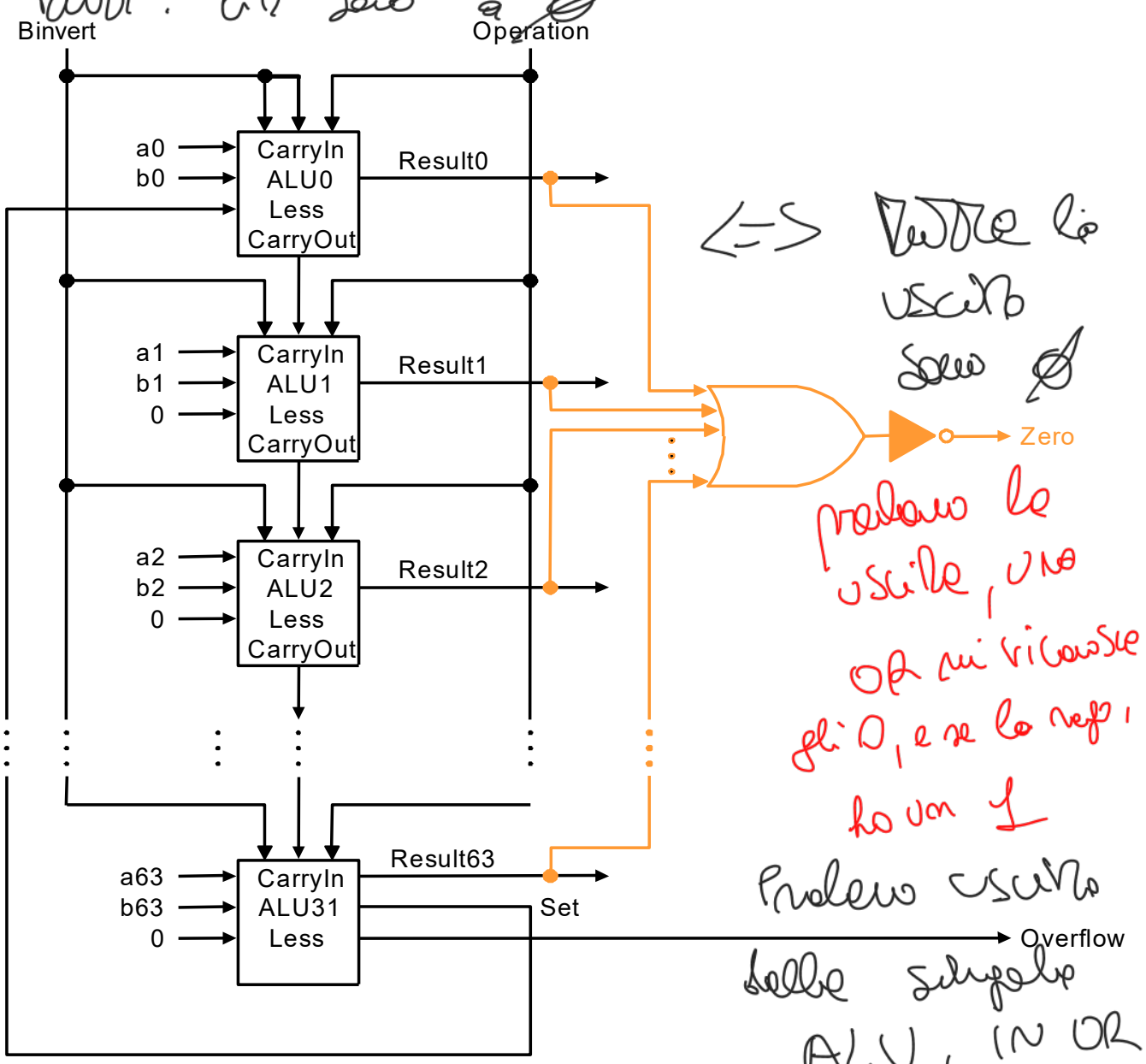
Operation	Function	CarryIn0	BInvert
0	AND	0	0
1	OR	0	0
2	ADD	0	0
2	SUB	1	1
3	SLT	1	1

→ $CarryIn_0$ = BInvert



Test per l'uguaglianza (eq)

- La rete fornisce il valore 1 quando tutti i bit del risultato sono a zero, in quanto: (a-b)==0 --> a==b
- Chiameremo tale segnale "Zero" e ci serve per decidere se l'esito del confronto è positivo o meno



Nota: "Zero" vale 1 quando il risultato è zero!

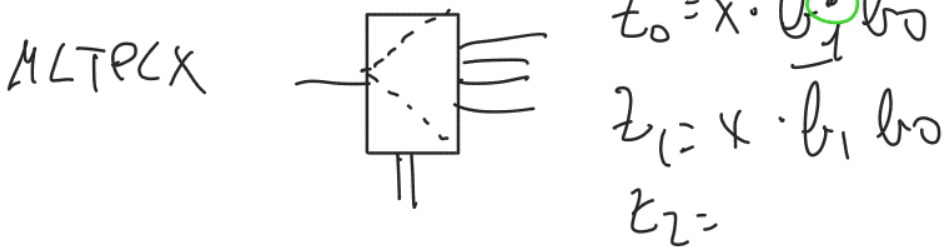
ALUcontrol

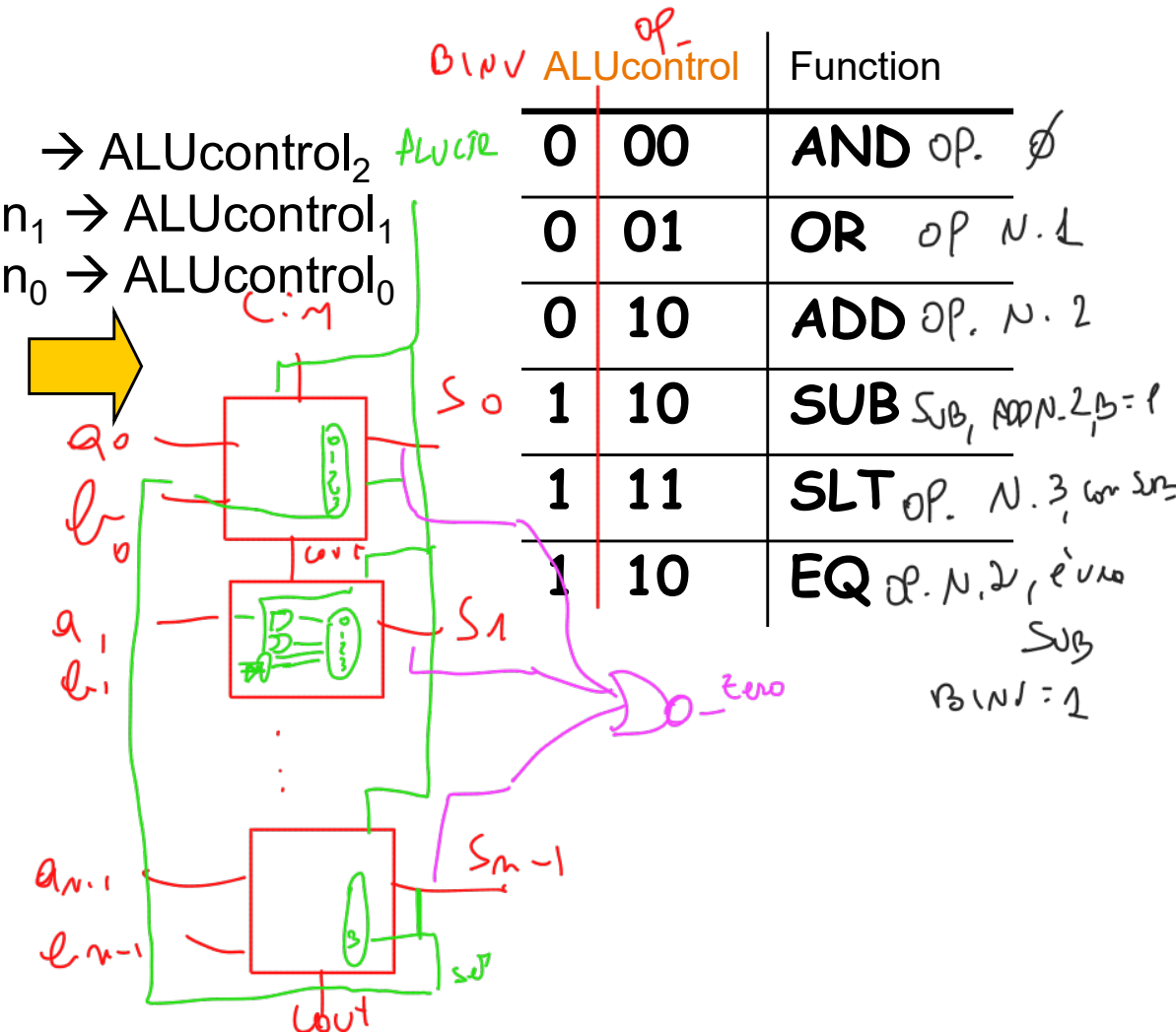
- Aggiungiamo quindi la funzione «EQ» che sfrutta la sottrazione e produce un ulteriore filo di uscita dalla ALU («Zero»)
- I fili di ingresso {Operation,Binvert} possono essere raggruppati nei fili chiamati «ALUcontrol» come mostrato in tabella

Operation	Function	BInvert
0 00	AND	0
1 01	OR	0
2 10	ADD	0
2 10	SUB	1
3 11	SLT	1
2 10	EQ	1

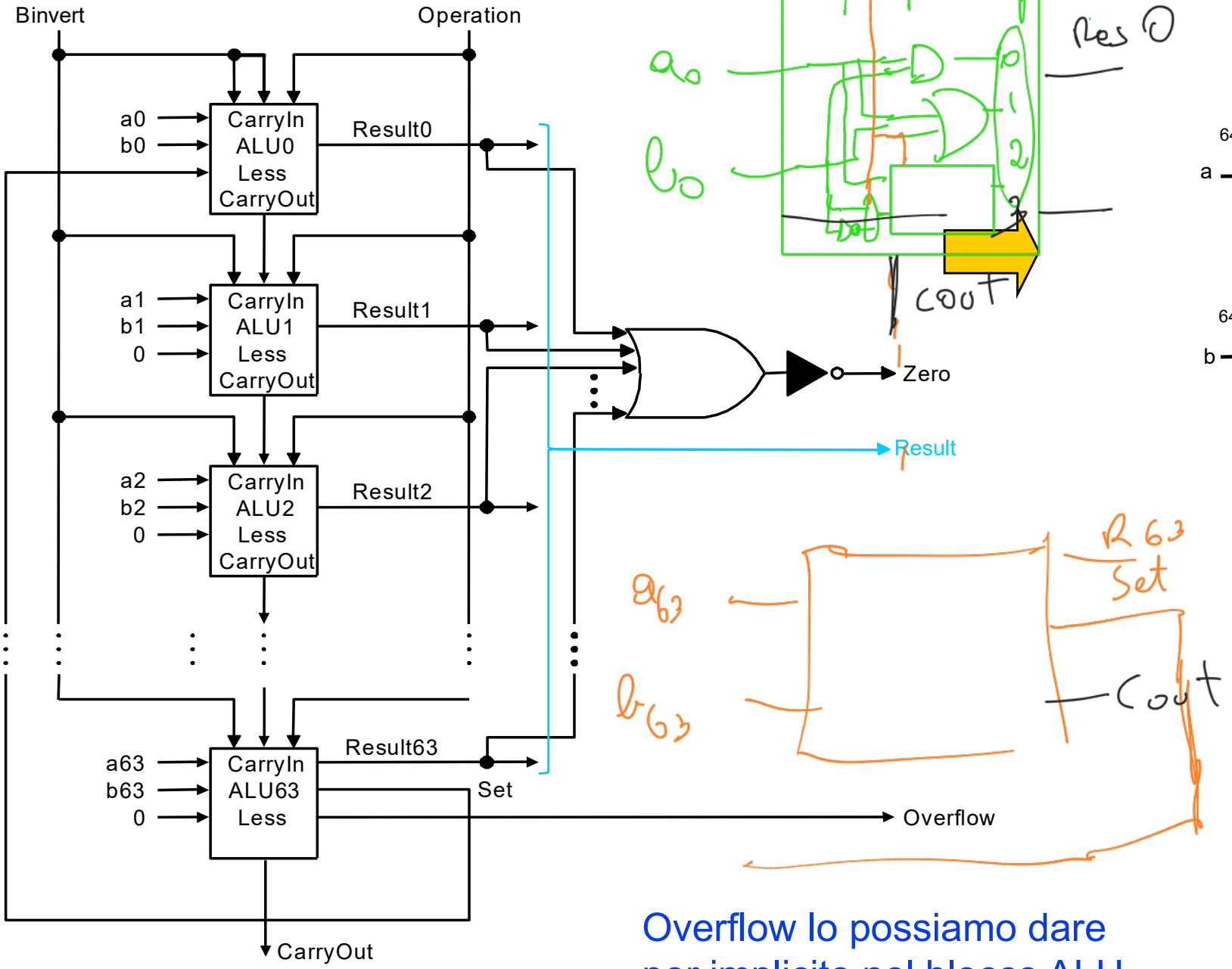
Binvert → $ALUcontrol_2$
$Operation_1$ → $ALUcontrol_1$
$Operation_0$ → $ALUcontrol_0$

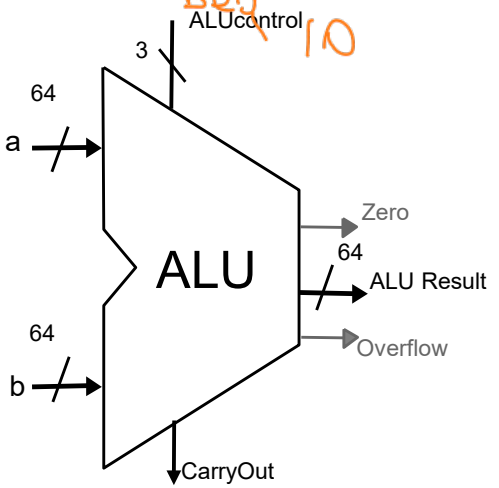
BINV	ALUcontrol	Function
0	00	AND
0	01	OR
0	10	ADD
1	10	SUB
1	11	SLT
1	10	EQ





Blocco "ALU a 64-bit" – situazione finale





Overflow lo possiamo dare per implicito nel blocco ALU