

Lezione 3

Elementi Architettureali Principali: Reti Combinatorie Notevoli

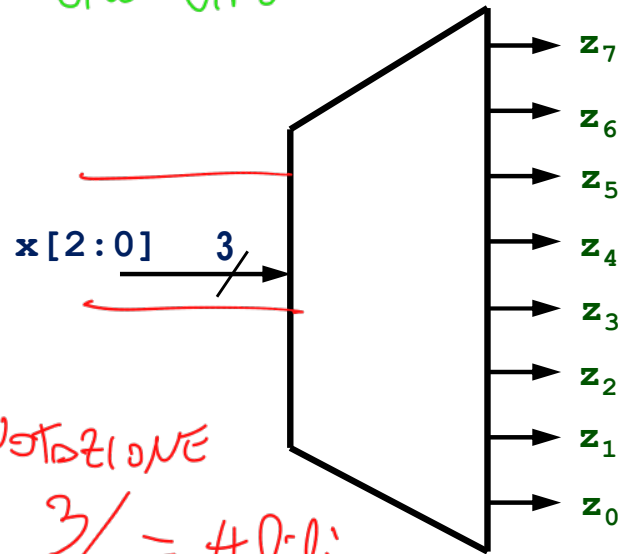
(v. PHRV1, appendice A.2, A.3, A.5, A.12)

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

Decoder

è una rete combinatoria, ci serve per decodificare dell'indirizzo
 in/out. 1/0, rappresentato con tabelle

riceva in ingresso n fili e sblocca
 uno uno solo dei 2^n fili in uscita



NOTAZIONE

$3/ = \# \text{ fili}$

raggruppo, bus

$\# \text{ Fili} = 3 \text{ in}, 2^3 = 8 \text{ out}$

in memoria

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

Tabella
vera'

$\text{in} = 101 = 5$

$\text{out} = z_5 = 1$

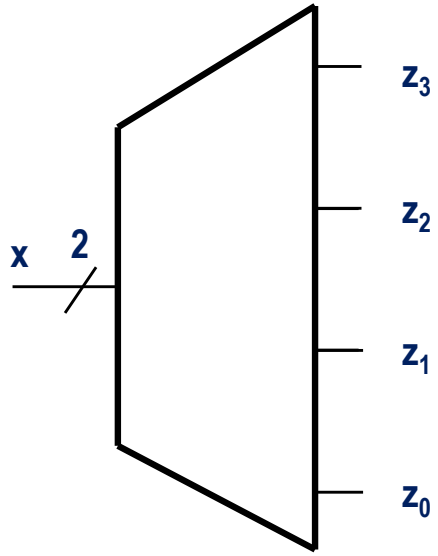
- Esempio di decoder a 3 bit o "da-3-a-8" riceve in ingresso n fili, e
- Un solo filo di uscita vale 1 (gli altri valgono tutti 0): quello di indice pari all'intero "posto sui fili" di ingresso
- In generale, un decoder da- n -a- 2^n ha n ingressi e 2^n uscite

sblocca solo
1 su 2^n fili.

Realizzazione del decoder

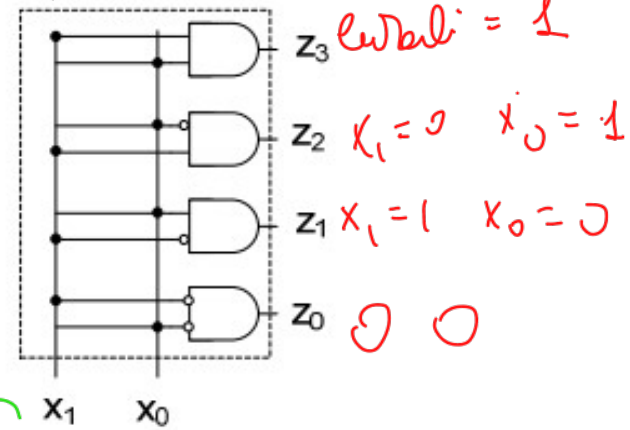
2 ingressi
da 2 a 4

- Si può 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

tutte e 4 varianti di porte and



porta AND in cui vedo a negare gli ingressi quando intendo sia 0, e li penso non negati quando intendo meno a 1

$z_3: x_1=1 \quad x_0=1$
 $z_2: x_1=0 \quad x_0=1$
 $z_1: x_1=1 \quad x_0=0$
 $z_0: 0 \quad 0$

Implemento con porte logiche, ma non è l'unica implemento. CMOS è più efficiente
da 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 efficiente 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

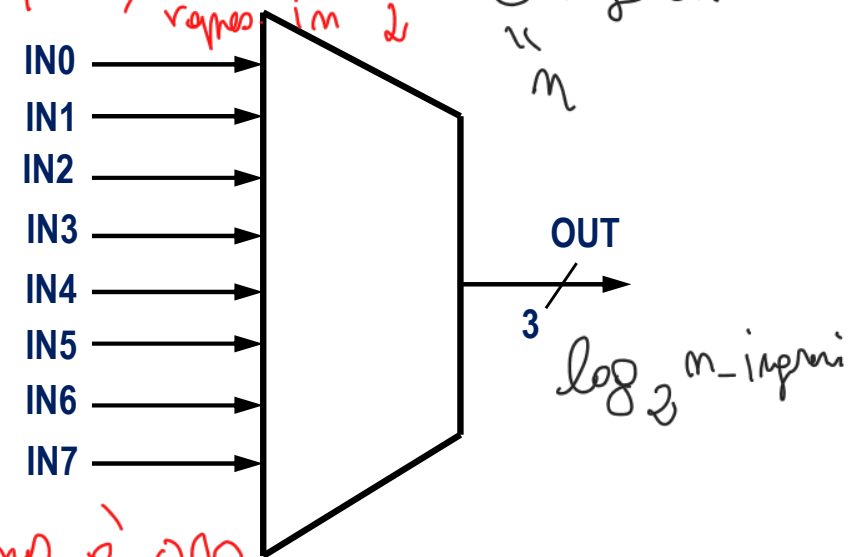
metto un AND in ingresso, e uscita vale solo se AND è 1 (?)

Encoder *due, righe il blocco, tutti ingressi e può usare* 2^n in, n out

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

Si accende un solo filo, in uscita ho rappresentazione in 2

$\# \text{ ingressi} = \log_2 n = 3 \text{ vs } n$



Solo uno è on
es. se sono in quella posizione
ATTIVO a quel cor.

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

stanno avere 2^n righe
ma n è una tabella di verità, stanno avere 256 righe
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")

ATTIVO IN5, in uscita ho ATTIVO 1 0 1
piccola rappres. di 5 in base 2

Realizzazione dell'encoder *es. Minterm* D_0

- $OUT0 = IN1 + IN3 + IN5 + IN7$

ho 4 ori

- $OUT1 = IN2 + IN3 + IN6 + IN7$

- $OUT2 = IN4 + IN5 + IN6 + IN7$

- 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 2ⁿ ingressi (gli altri sono posti a 0):
le n uscite producono i bit corrispondenti alla rappresentazione in base due di k

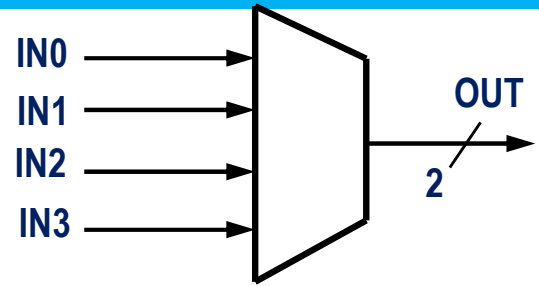


TABELLA DI VERITÀ (COMPLETA):

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

FUNZ.

IN1, IN0 IN3, IN2	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

IN1, IN0 IN3, IN2	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

e 1/0? non serve se 1/0 vale 0, sono gli altri che determinano uscita

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

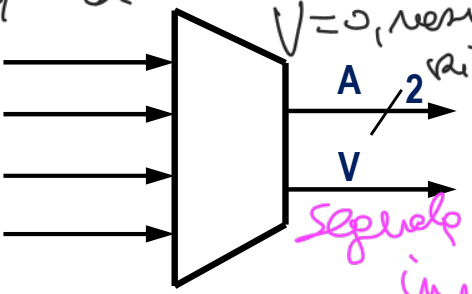
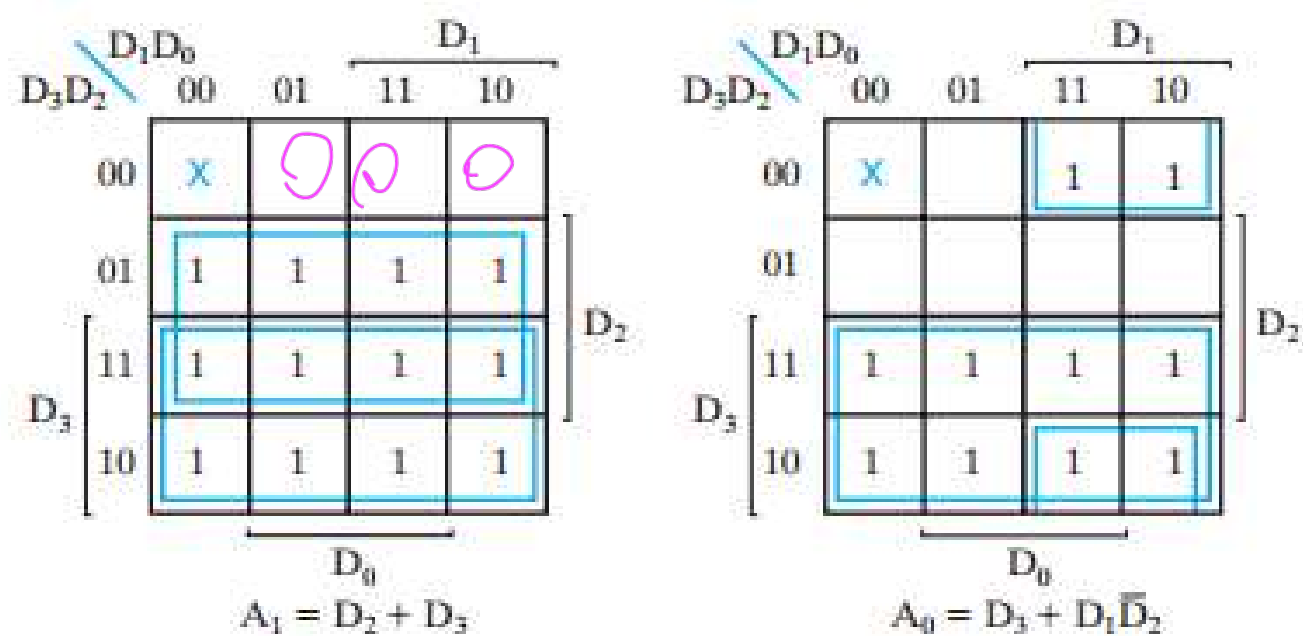


Tabella di verità completa
ho tutti i casi

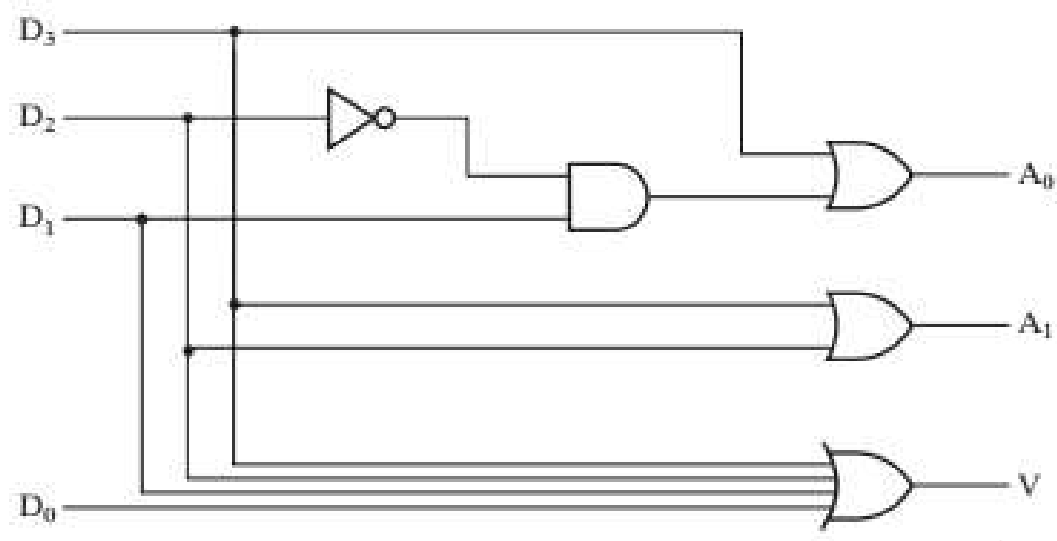
- Quando D₃ vale 1 tutti gli altri bit non contano e l'uscita indica la massima priorità (11)₂ ovvero 3
- Quando D₃ vale 0 ma D₂ vale 1 gli altri bit a destra non contano e l'uscita indica priorità 2 e così via...
- Il bit V è necessario per indicare se l'uscita è valida ovvero se almeno un 1 è stato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



Usare valore x i se
 la rappresentazione di B vale i

Serve per INSERIRE UNO DEI TANTI ING. IN USCITA

Multiplexer

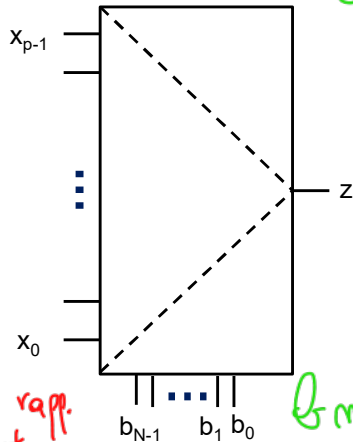
Ingressi convogliati: in un unico filo, introduce sul filo di uscita

- Il multiplexer serve per selezionare un'informazione

quale segnale prende per introdurla nell'uscita. In ingresso ho p fili, in uscita ho un solo filo

- 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

uscita = ingresso; se rappresento di $b = i$



B, intero rapp. su m bit

Multiplexer con 2^N ingressi

B mi dice quale dei due 2^N fili va nell'uscita

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

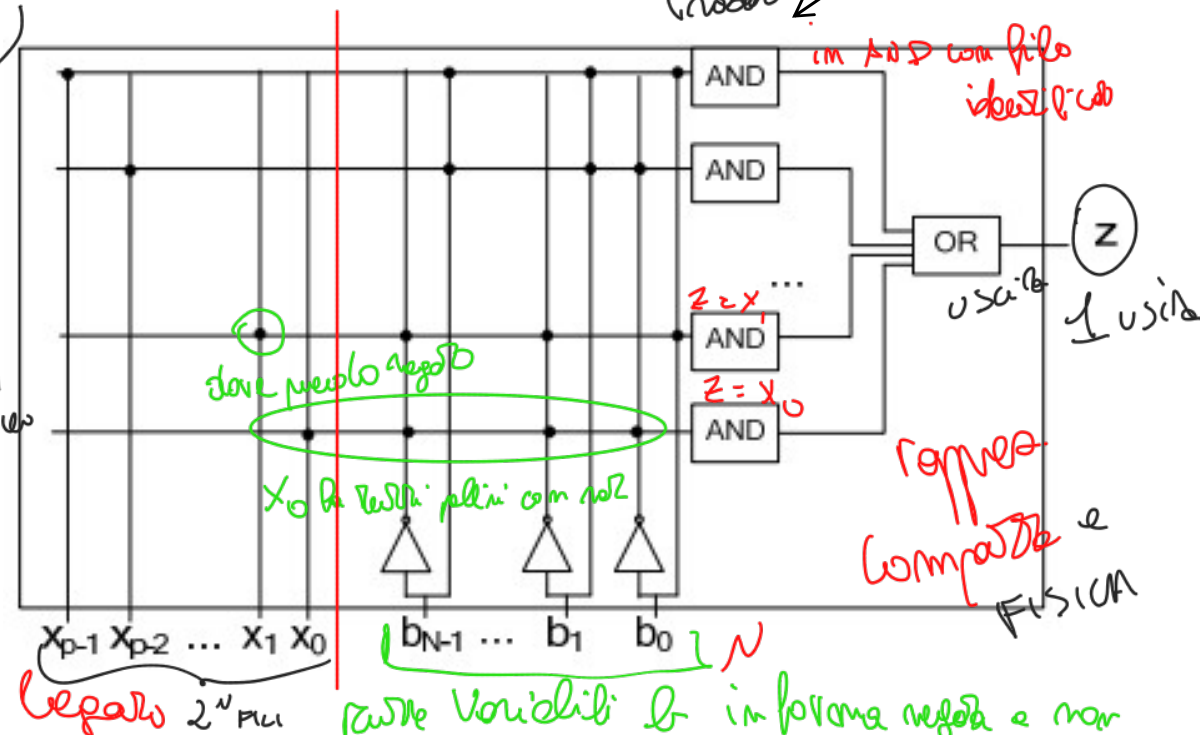
AND con N+1 ingressi

Enable

in AND con filo invertito

uscita 1 uscita

rappr. completa e FISICA



dove prende segnale

x_0 ha tutti i fili con not

MINTERM che identif. uno tra gli N cor.

ingreso legato 2^N fili

parte variabili b informazione negata e non

MINTERM

tutti i var.

$$z = x_0 \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot b_0 \oplus$$

$$x_1 \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot b_0 \oplus$$

$$x_{p-2} \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot b_0 \oplus$$

$$x_{p-1} \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot b_0$$

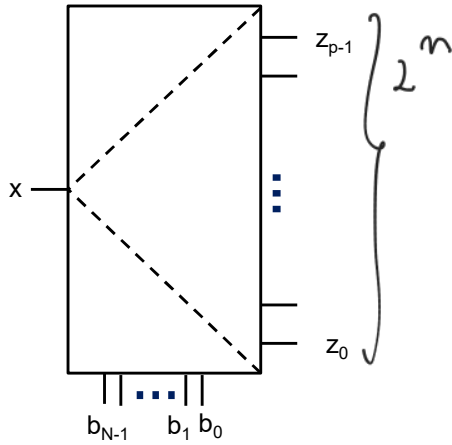
e li metto in corrispondenza con l'ingresso legato

Demultiplexer

le variabili di controllo

viene SELEZIONATO 1 dei 2^N FILI

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



- 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

Demultiplexer con 2^N uscite

P uscite, P equas. Ho un insieme di eq

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

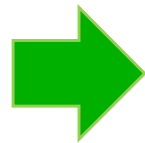
$$z_1 = x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot b_0$$

...

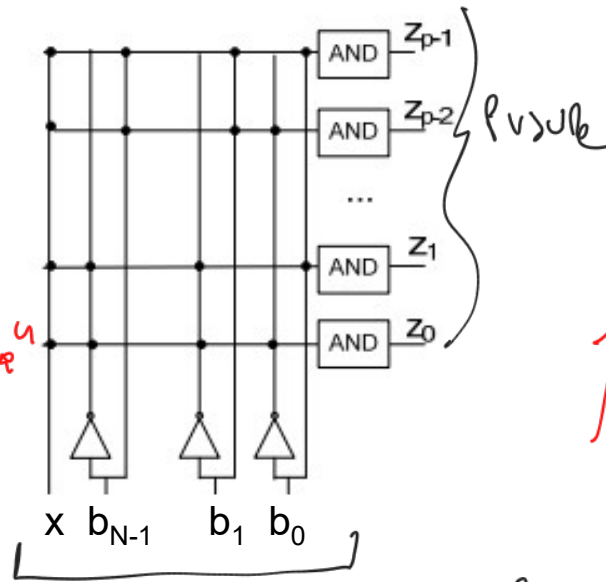
$$z_{p-2} = x \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot \overline{b_0}$$

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

ho p equas. ex



come prima



dato un filo, 2^N variabili uscite, N variabili controllo

Viene selezionata l'uscita corrispondente al controllo e le altre uscite sono 0

Può essere visto come un decoder con abilitazione

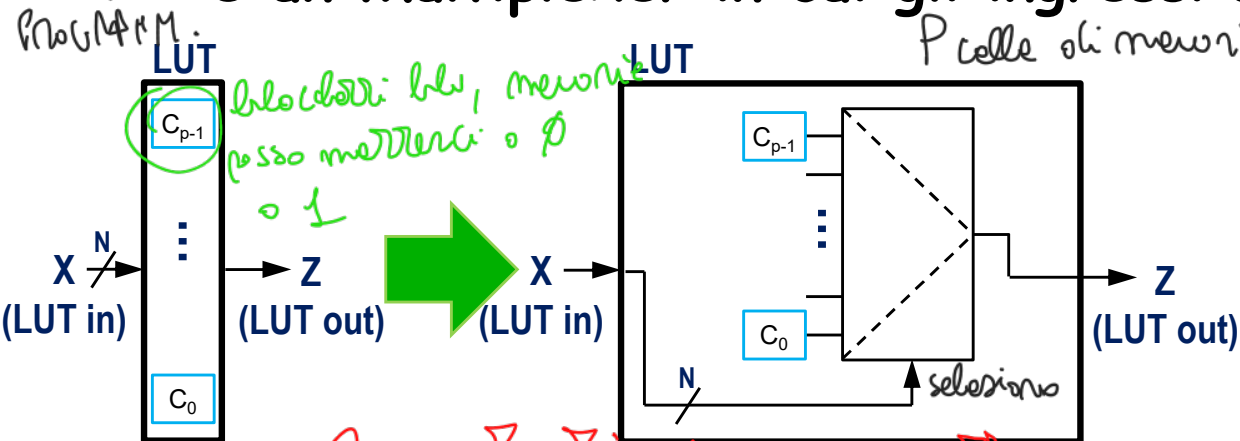
Stesso blocco per selezionare, P uscite

Look Up Table (LUT)

- ho p celle di memoria da 1 bit (v. PHRV1, appendice A.12)
- con X (meno 2 di N bit) vedo a selezionare quale delle 2^N

MULTIPLEXER: è un Multiplexer in cui gli ingressi sono costanti

Celle mi interessano in uscita



- 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)

- C costanti, se pero mettiamo posso programmare con una LUT, una qual'sos. F. booleana a N bit
- ~~è utile per realizzare una funzione booleana generica~~

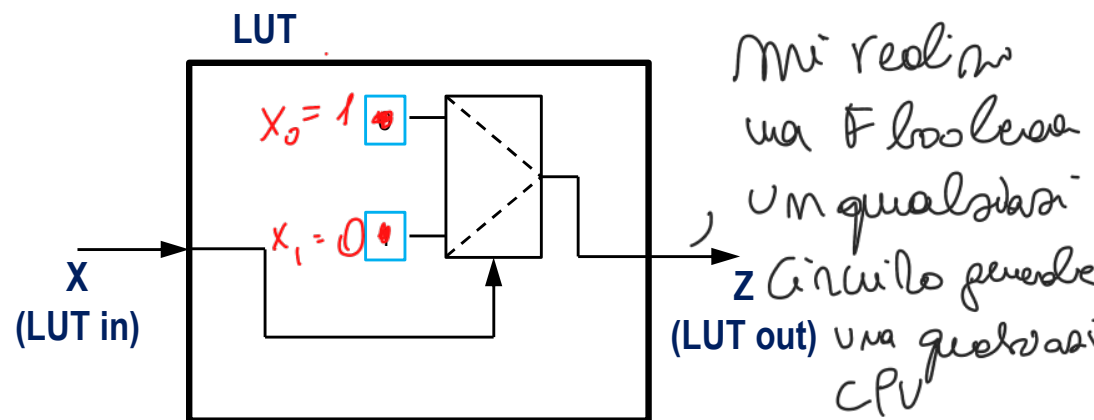
C, valori che mi aspetto in uscita data

- Esempio realizzazione della funzione "inverter" tramite LUT

X	Z
0	1
1	0

Profondità (depth) *Idella*

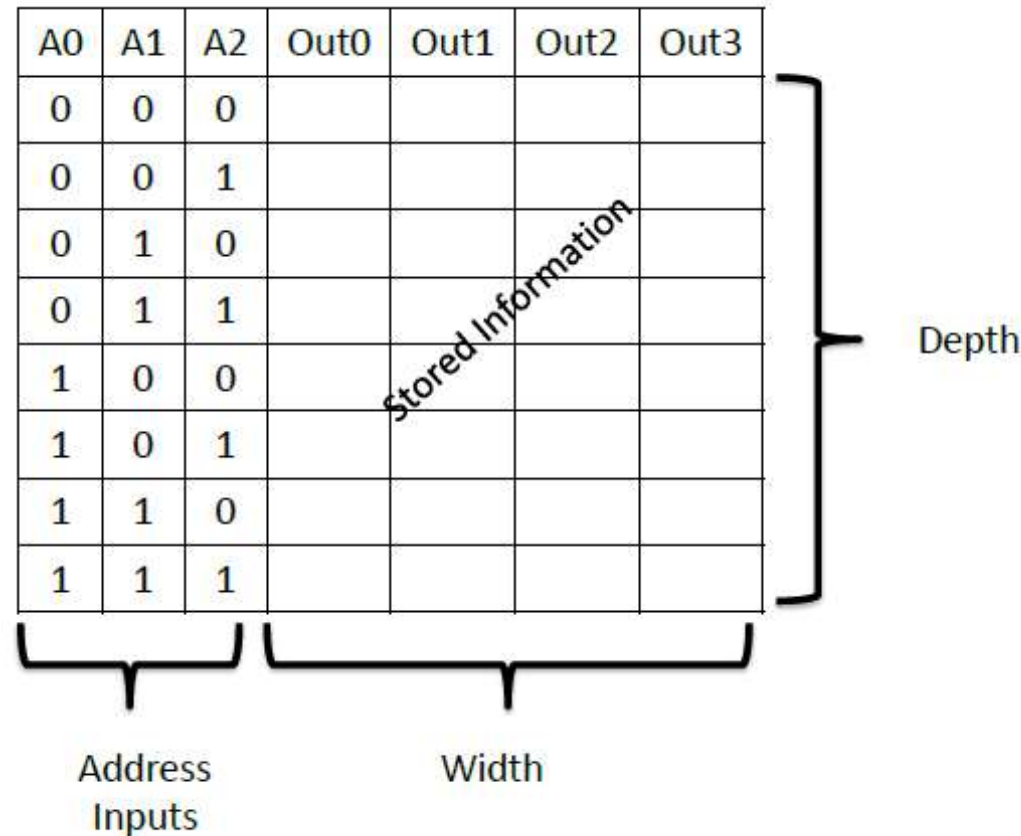
Ingressi Larghezza (width)



ho N var. di ingresso, memorizzo nella C i valori che mi aspetto in uscita e realizzo così

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)

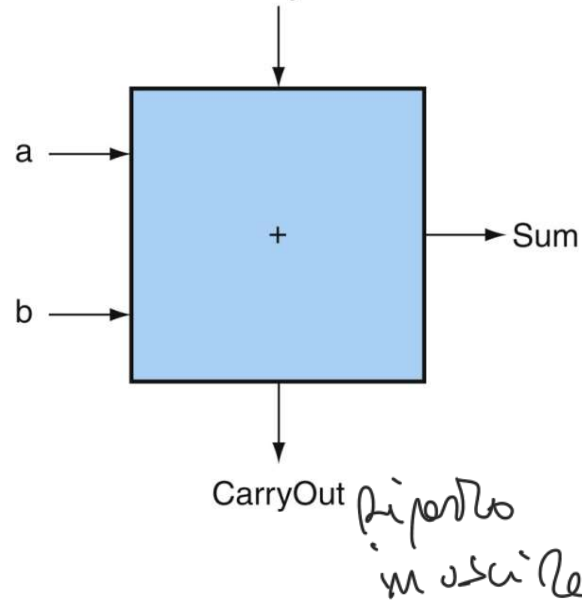
Curve del processore

ALU esegue di un processore,
ha le 4 operazioni: somma, sottr., AND
e OR

Full Adder a 1 bit

circolo che fa la somma di 2 bit a e b

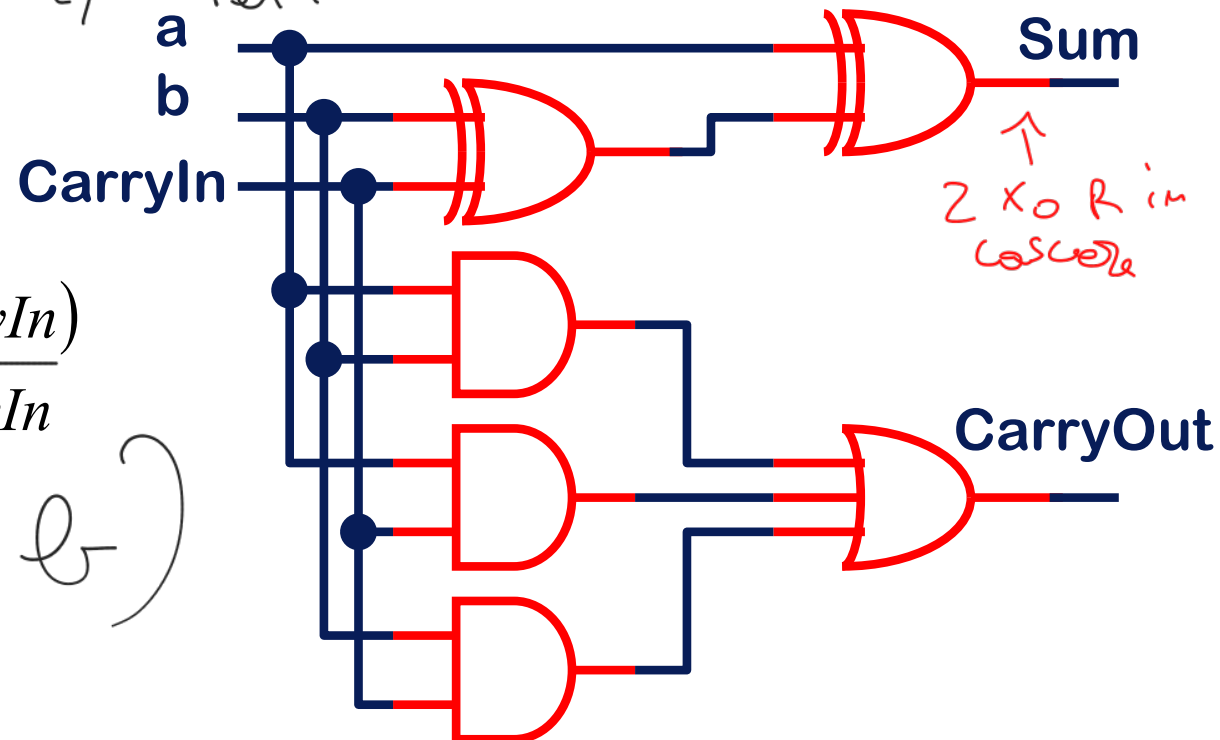
Neto continuazione CarryIn



Riparto in ingressi

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

2 uscite, 2 reti



Somma = XOR INGRESSI

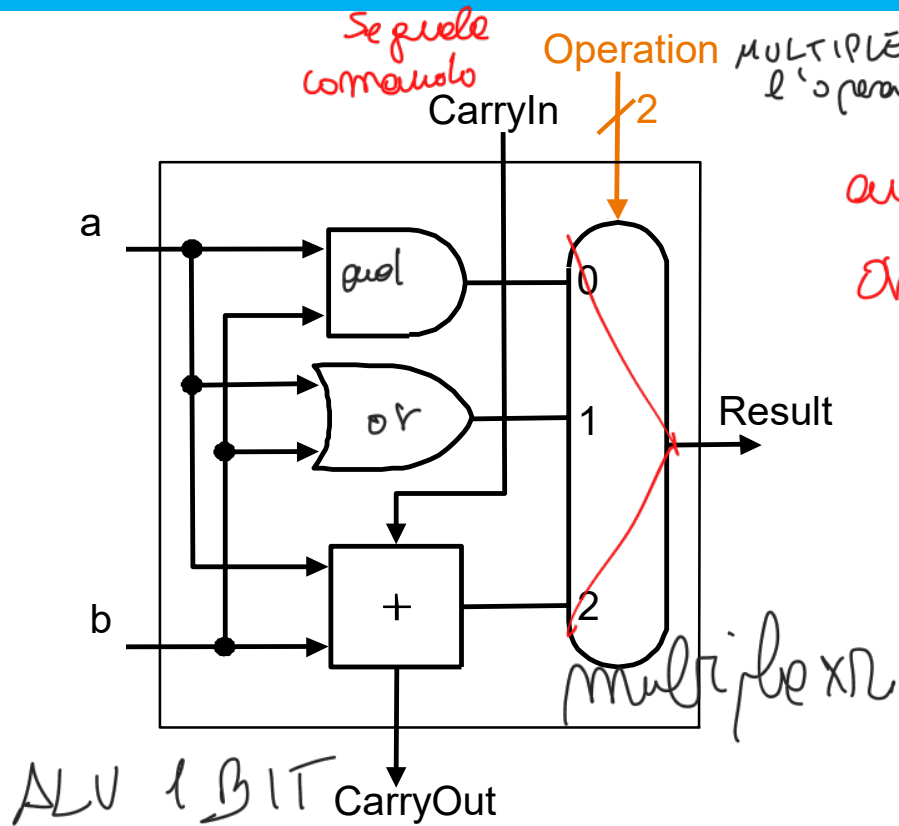
$$\text{sum} = a \oplus b \oplus \text{CarryIn} = a \oplus (b \oplus \text{CarryIn})$$

$$\text{CarryOut} = a \cdot b + a \cdot \text{CarryIn} + b \cdot \text{CarryIn}$$

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

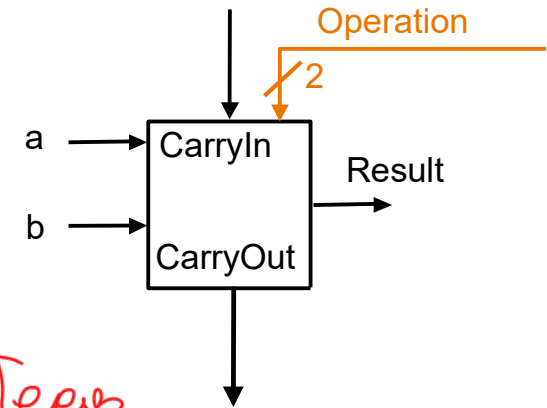
Semplice ALU a 1-bit

AND = operazione 0 OR = operazione 1 ADD = operazione 2



Operation	Function
0	AND
1	OR
2	ADD

ALU a 1-bit

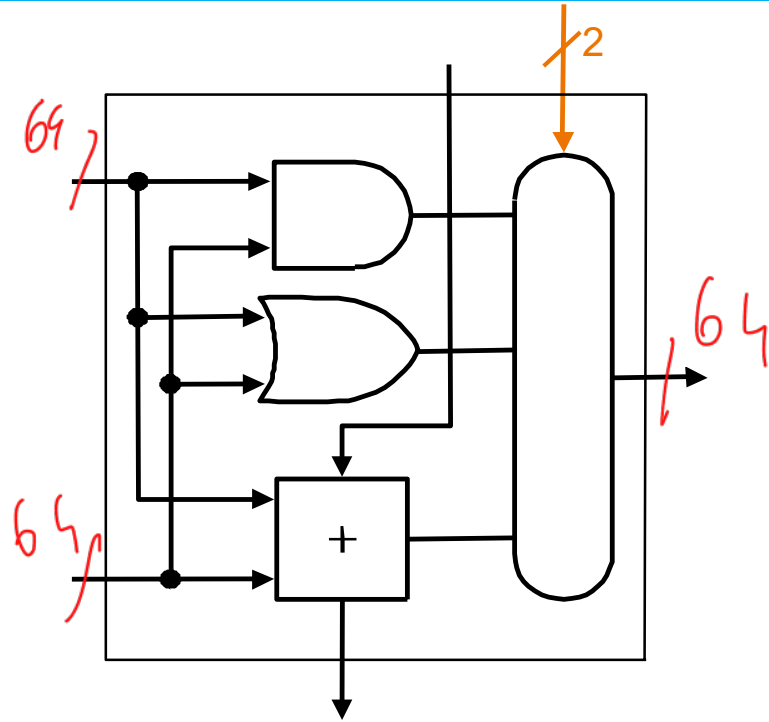


Full Adder = Tees comb del v:ports

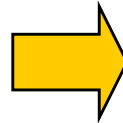
operation = sequela di comando, mi' seleziona l'operazione

Costruire una ALU a 64 bit

Usando in parallelo ad ogni ALU elementare
l'operazione da svolgere, anzi in uscita
i 64 bit in uscita



da 1 a 64 bit

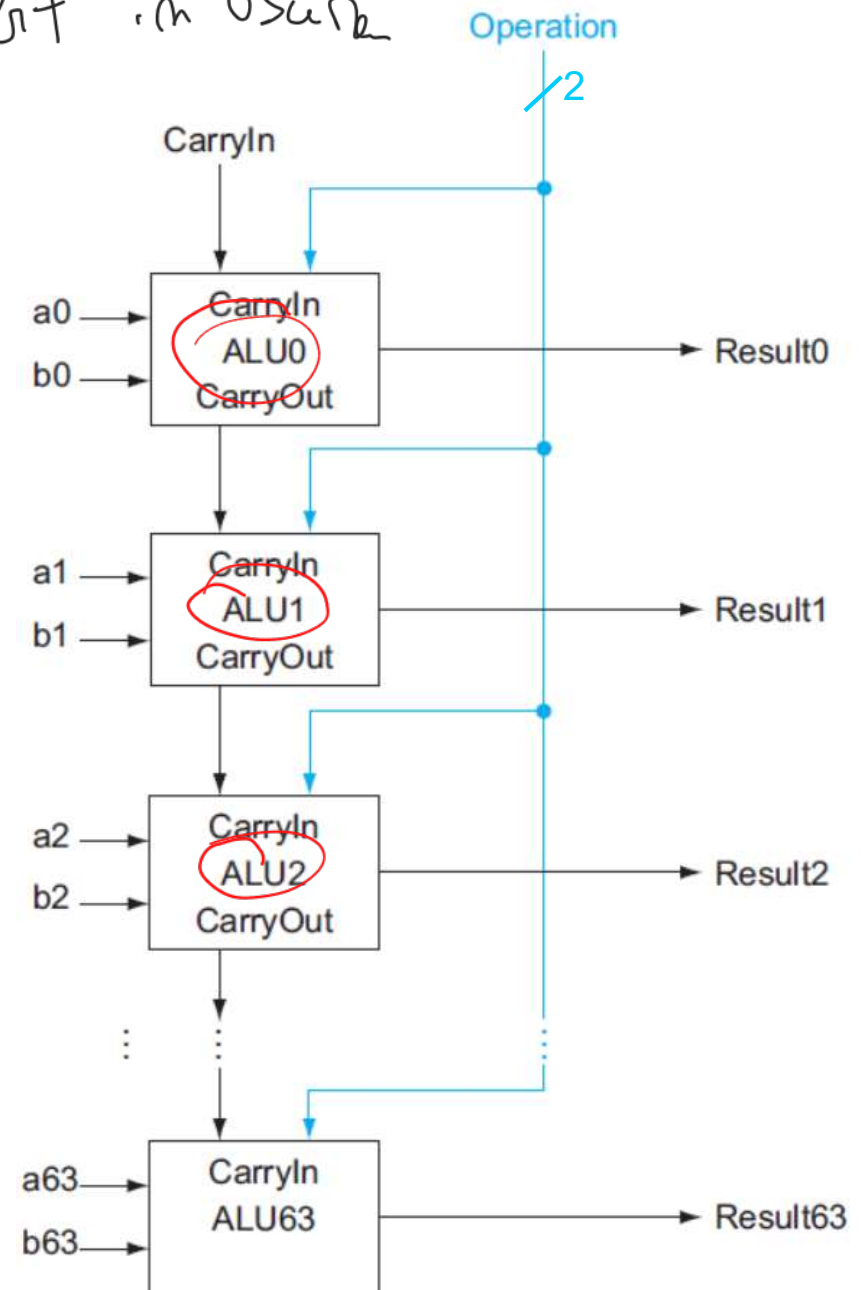


ad ALU ϕ
ho come ingresso
i bit a_0 e b_0

Un sommatore costruito in questo modo è chiamato
“sommatore con riporto seriale”

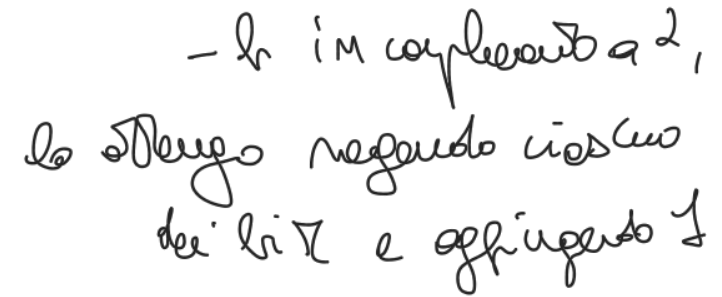
Problema: *non e' efficiente*
il risultato è disponibile solo dopo che ALU63
ha ricevuto il carry

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



per $\bar{b}$ inserimento a

- $$a + b \text{ nepo} \approx nb + 1$$



SODAS = Summa
 Com Einwert = 1
 neg 6

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

Per vedere se, vedo bit # negl. $\rightarrow$ $(xxxxx$
bit significativi, che in complement.
 $a - b = 1$

Se ALU 63 ha bit di uscita = 1,
numero è negativo.

$\rightarrow$ ripeto e' 1
in uscita, esito oper.

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 $a = b$

uso sottrazione.

vedo se il
numero $a-b$ per ultimo
bit negativo

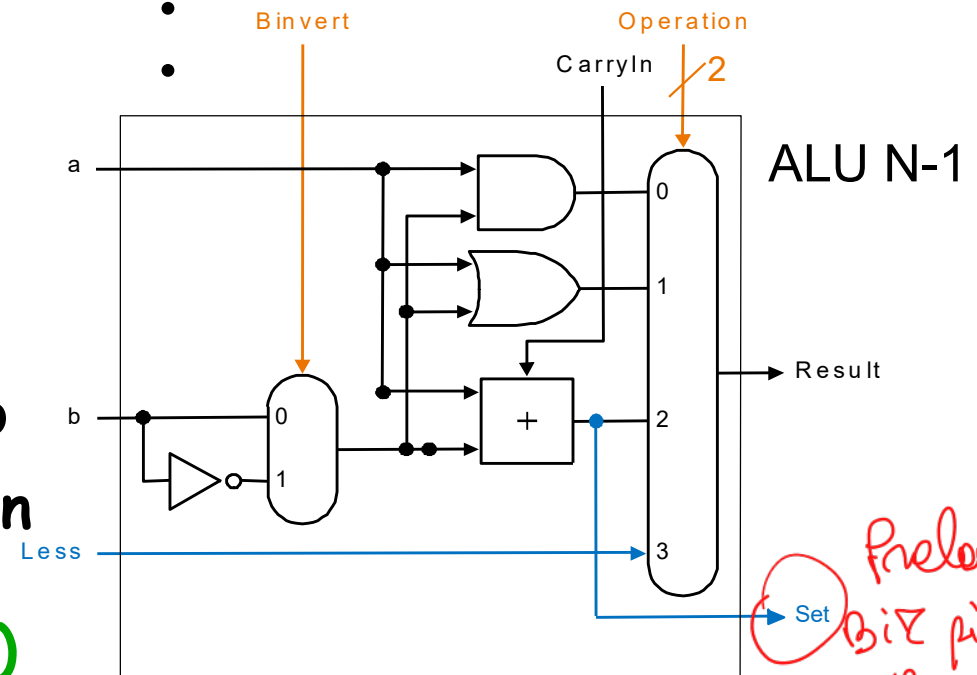
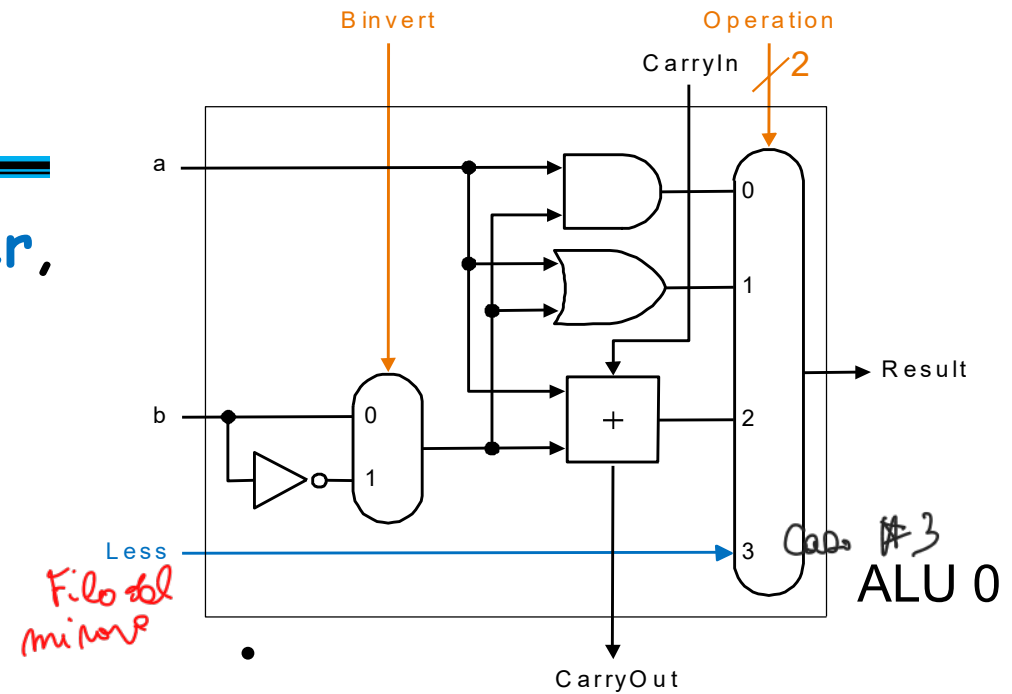
Se bit 63 = 1, $a < b$
ripeto il bit sull'uscita
e metto per bit
bit a 0

$\rightarrow$ In entrambi i casi si può sfruttare la rete per fare la sottrazione

uso in entrambi i casi la stessa rete
usata per la sottrazione.

Supporto alla slt nella ALU

- Inserisco il caso "3" al multiplexer, corrispondente a slt *passa*
- 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 $a < b$ *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 *no bisogno di una rete logica un poco più complessa (v. slide finale)*



Primo bit più significativo

ho allora 3 fili di comando (ma Binvert = CarryIn)

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 CarryIn₀:

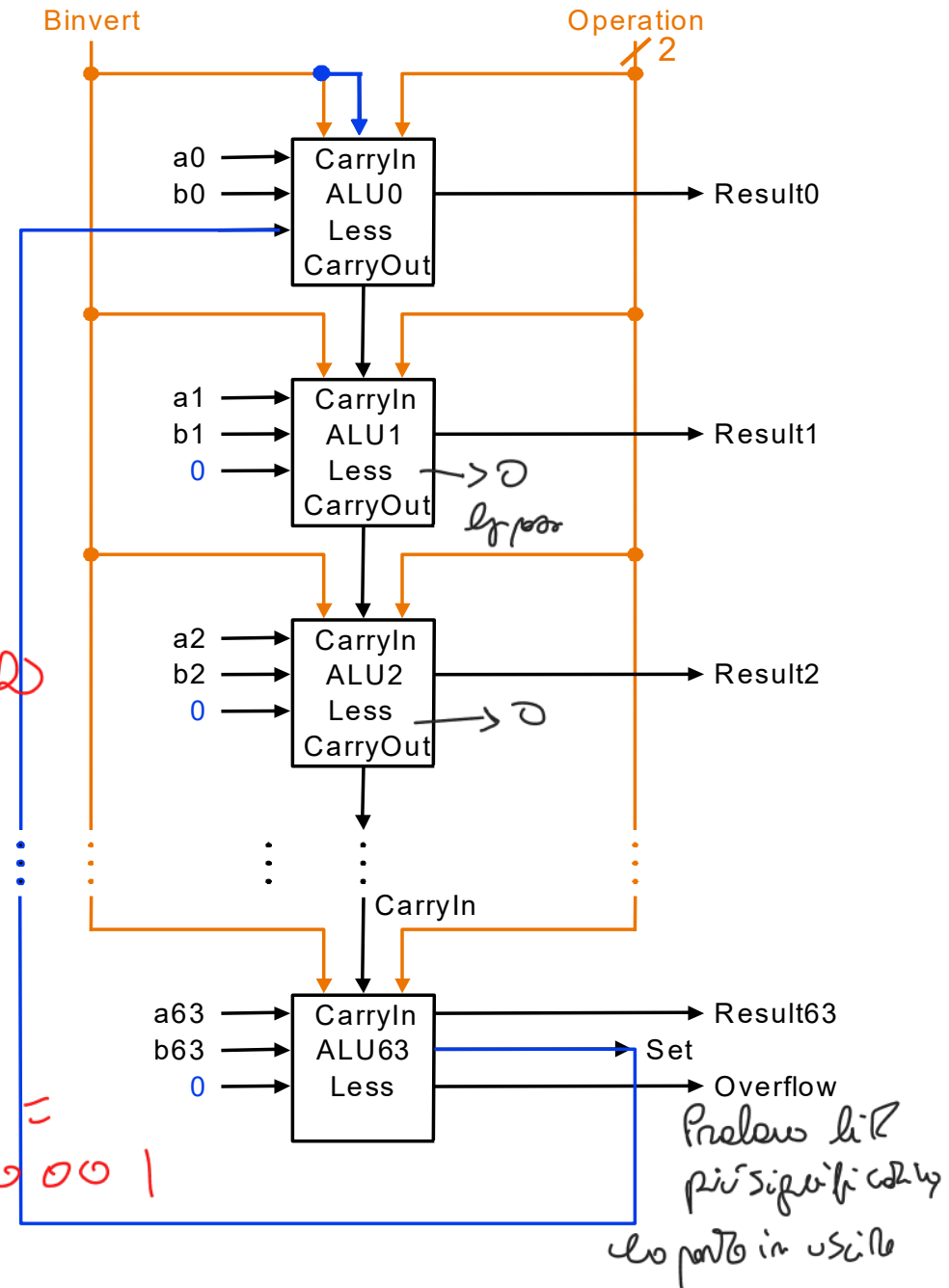
sono uguali.

Operation	Function	CarryIn ₀	Binvert
0	AND	0	0
1	OR	0	0
2	ADD	0	0
2	SUB	1	1
3	SLT	1	1

per settare.

→ CarryIn₀ = Binvert

Usa la sara LESS = 00000001



Test per l'uguaglianza (eq)

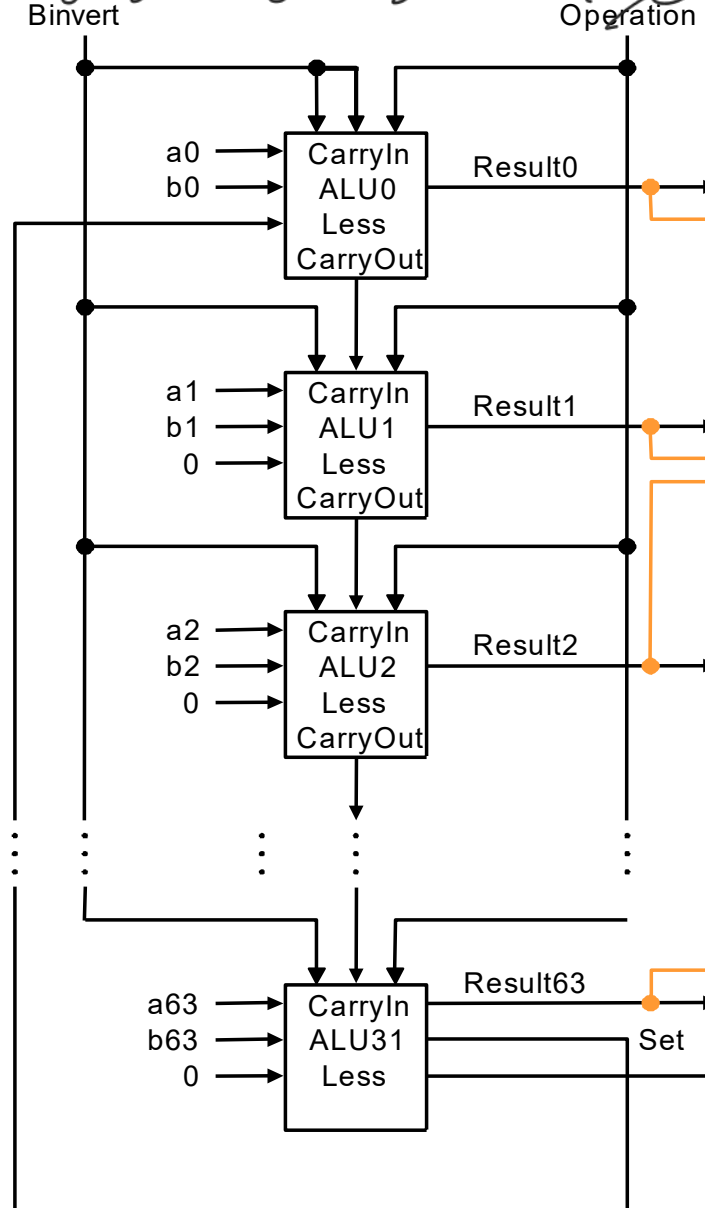
riconoscere $\emptyset$ in uscita

SOTTINIZ, vedo se tutti i bit sono a zero

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

porta OR in uscita
 riconosce tutti $\emptyset$,
 la NEG

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



$\Leftarrow$ tutte le uscite sono $\emptyset$

preleva le uscite, una op mi riconosce gli 0, e se lo nego ho un 1

Preleva uscite dalle singole ALU, in OR con inverter

ALUcontrol

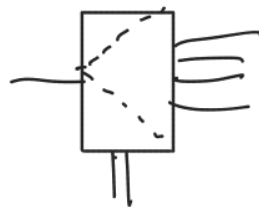
- Aggiungiamo quindi la funzione «EQ» che sfrutta la sottrazione e produce un ulteriore filo di uscita dalla ALU («Zero»)
- I fili di ingresso {Operation, Binvert} possono essere raggruppati nei fili chiamati «ALUcontrol» come mostrato in tabella

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 $\rightarrow$ ALUcontrol₂
 Operation₁ $\rightarrow$ ALUcontrol₁
 Operation₀ $\rightarrow$ ALUcontrol₀

Binvert	ALUcontrol	Function
0	00	AND op. $\emptyset$
0	01	OR op. N.1
0	10	ADD op. N.2
1	10	SUB $SUB, ADD, N.2, B=1$
1	11	SLT op. N.3, $with\ sign$
1	10	EQ op. N.2, $è\ una\ SUB\ Binvert=1$

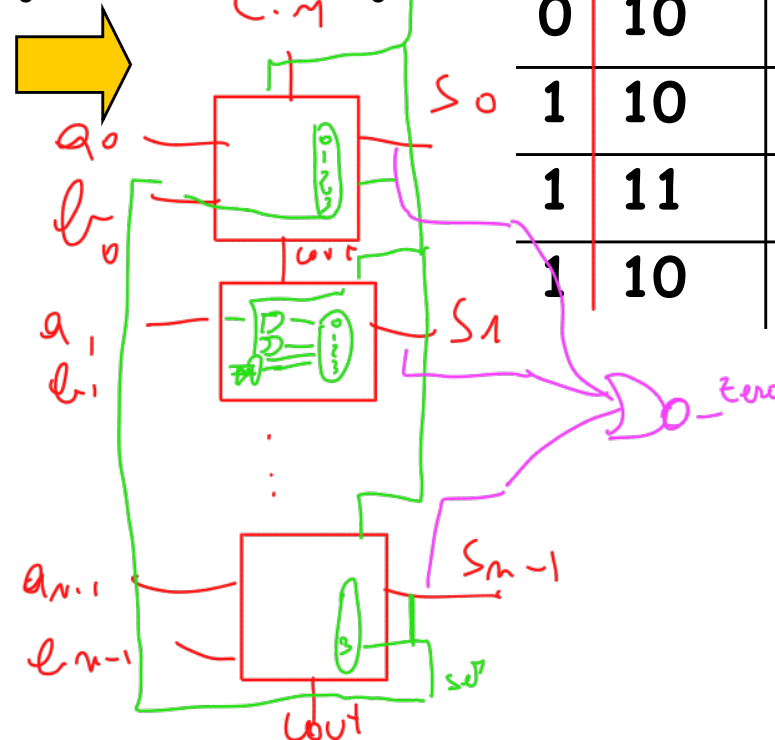
ALTPCX



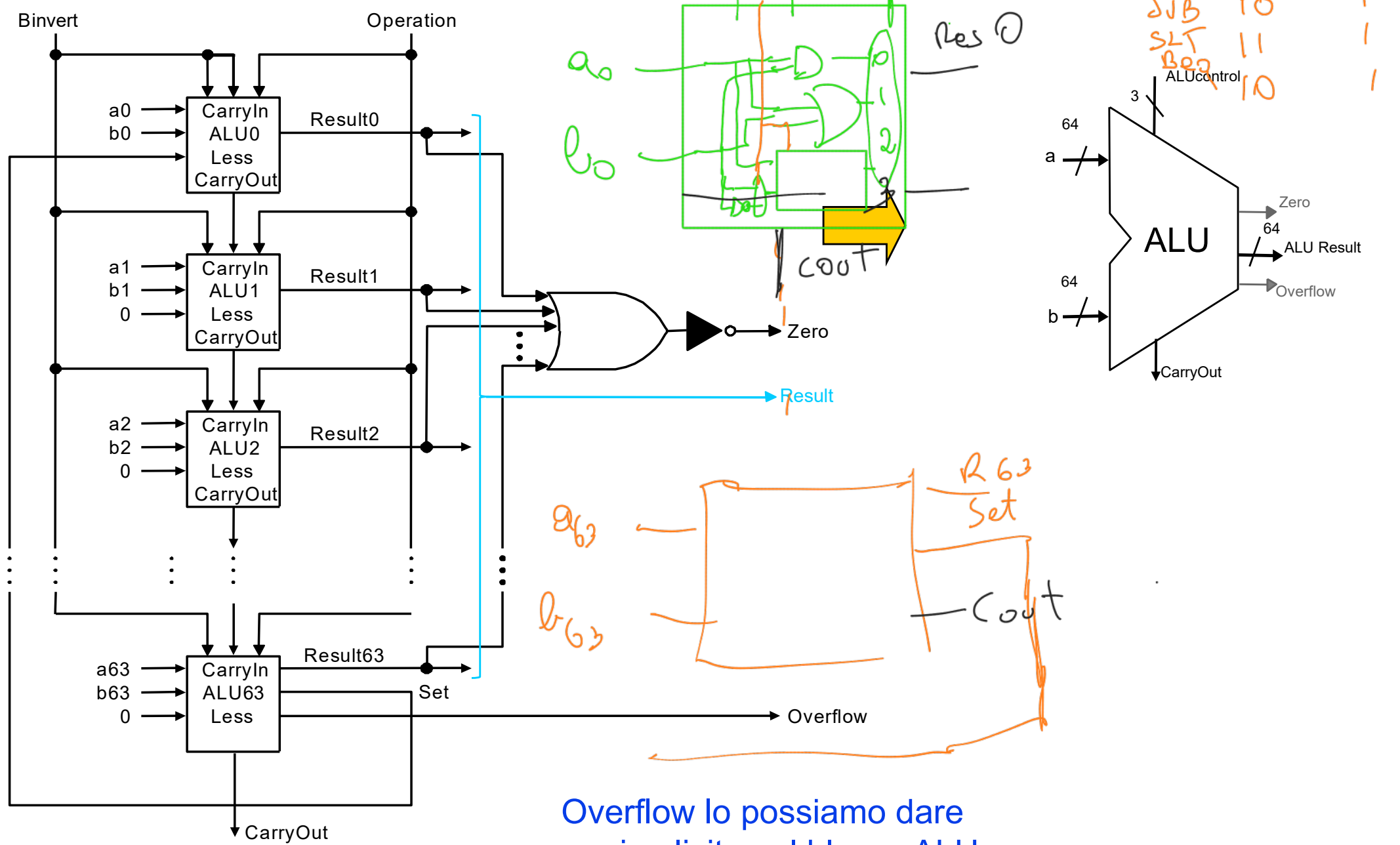
$$z_0 = x \cdot b_0$$

$$z_1 = x \cdot b_1$$

$$z_2 =$$



Blocco "ALU a 64-bit" - situazione finale



Overflow lo possiamo dare per implicito nel blocco ALU