UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

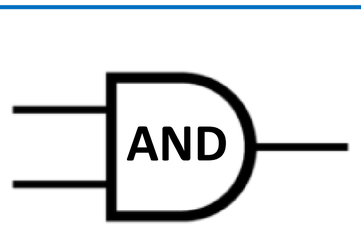
Circuiti in logica combinatoria e ALU

Prof.ssa Giulia Cisotto

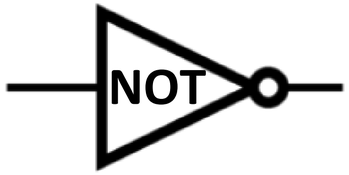
giulia.cisotto@units.it

Trieste, 24 marzo 2026

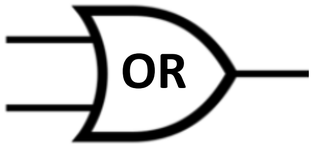
RECAP



A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

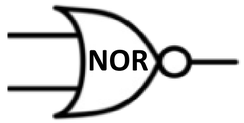


A	$\overline{A}$
0	1
1	0

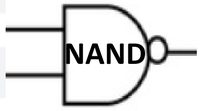


A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

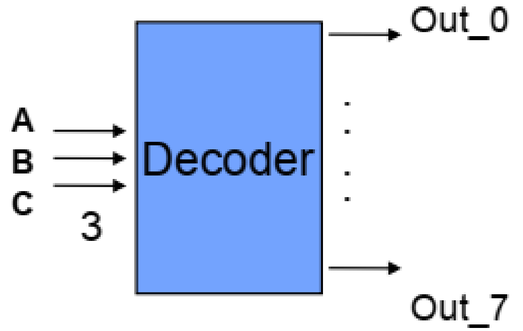
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

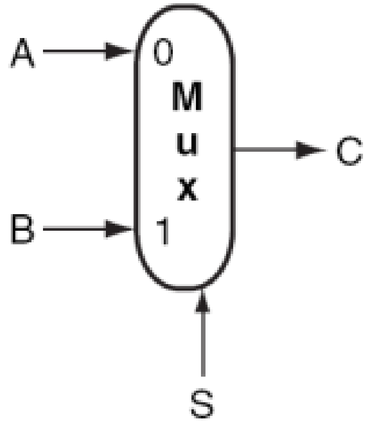


A	B	$A + B$	A NOR B
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0



A	B	$A \cdot B$	A NAND B
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

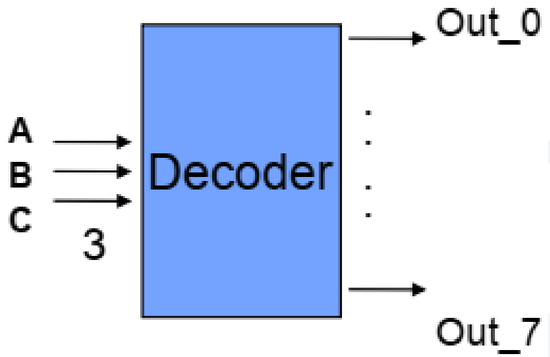




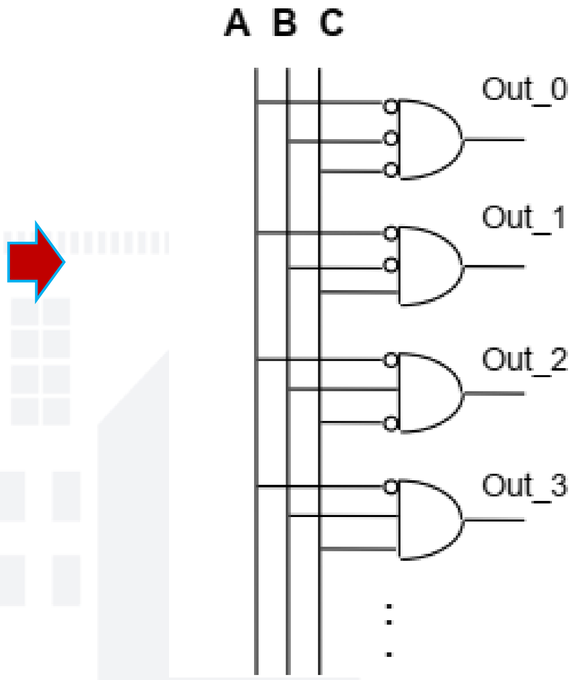
UNIVERSITÀ DEGLI STUDI DI TRIESTE

DECODER

Il decoder riceve in **input b segnali** ed è in grado di **selezionare UNA tra 2^b uscite**.

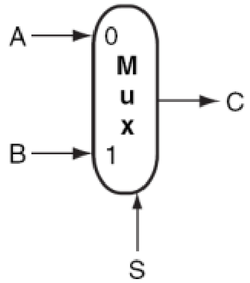


A	B	C	0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Si seleziona UNA uscita.

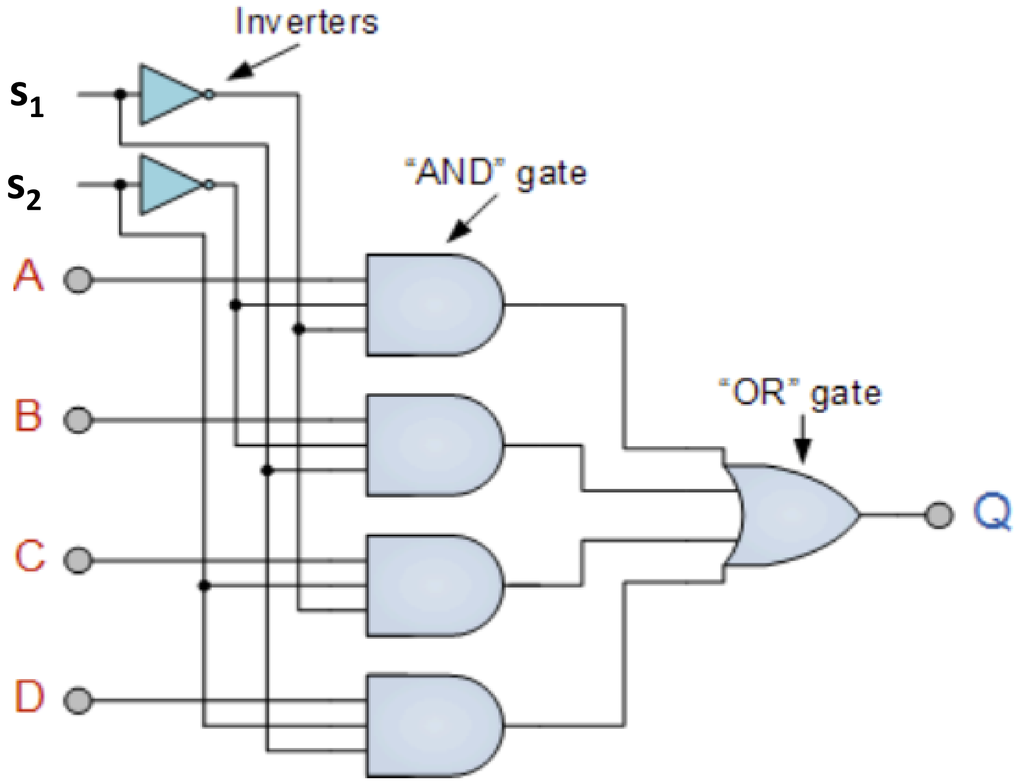
MULTIPLEXER



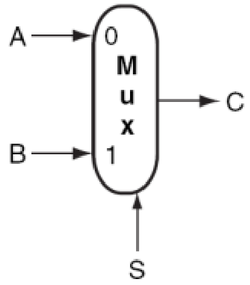
Se un multiplexer riceve in **input n segnali**, esso necessiterà di **$\log_2 n$ selettori (=# bit del segnale di controllo)** e consisterà di:

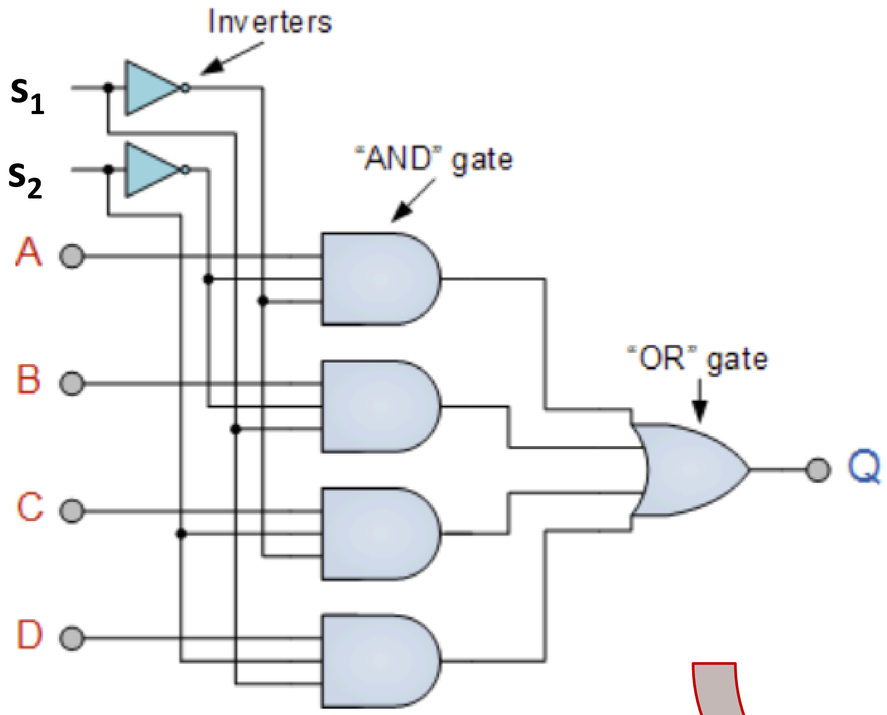
$$n = 2^m \quad \longleftrightarrow \quad m = lg_2\, n$$

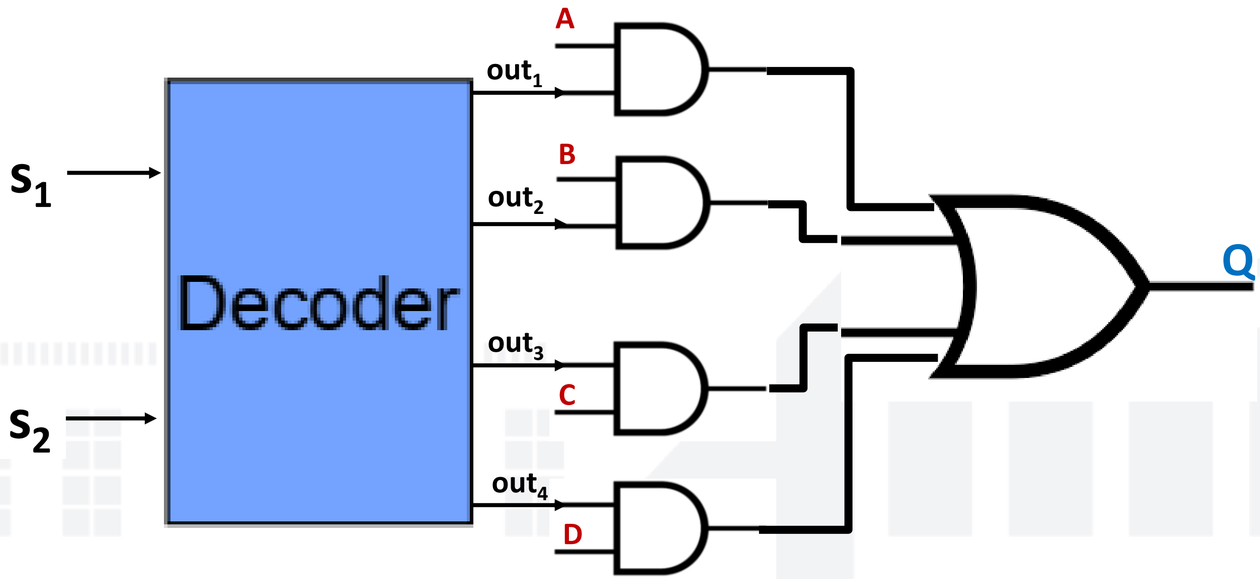
1. Un decoder che genererà n segnali
2. Un array di n porte logiche AND
3. Un'unica porta logica OR



MULTIPLEXER (tramite decoder)







Si usano (apparentemente) più elementi circuitali!

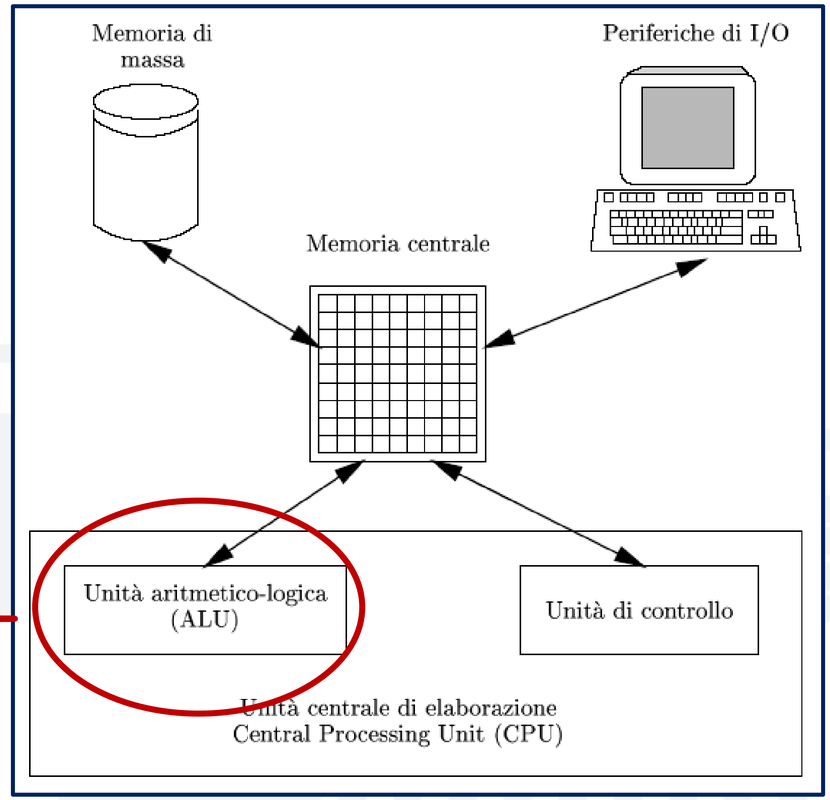
Perché usare questa implementazione quindi?

Vantaggi:

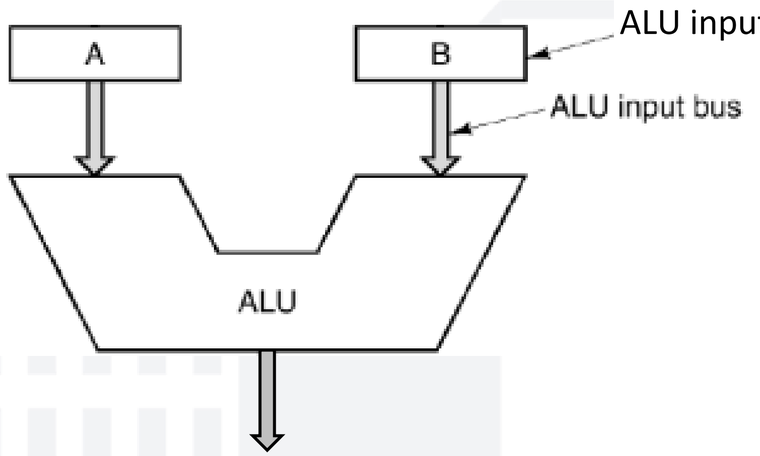
- possibile riuso del decoder
- migliore modularità
- migliore scalabilità
- migliore fattibilità

RECAP

```
add $t0, $t1, $t2
and $t0, $t1, $t2
or $t0, $t1, $t2
xor $t0, $t1, $t2
sll $t0, $t1, 2
      :
```



Per svolgere operazioni aritmetico-logiche «bastano» **CIRCUITI LOGICI COMBINATORI**, ovvero non ci serve memorizzare nulla (come nei circuiti logici sequenziali).



UNIVERSITÀ DEGLI STUDI DI TRIESTE

ARITHMETIC LOGIC UNIT (ALU)

L’Aritmethic Logic Unit:

- E’ la parte del processore che svolge le operazioni aritmetico-logiche
- E’ un insieme di circuiti combinatori che implementa:
  - Operazioni aritmetiche: es somma e sottrazione
  - Operazioni logiche: es AND e OR

BLOCCHI DI BASE PER COSTRUIRE L'ALU

1. AND gate ($c = a \cdot b$)

a	b	$c = a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

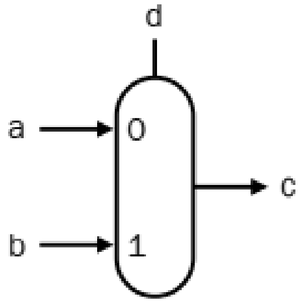
2. OR gate ($c = a + b$)

a	b	$c = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

3. Inverter ($c = \bar{a}$)

a	$c = \bar{a}$
0	1
1	0

4. Multiplexor
(if d = = 0, c = a;
else c = b)



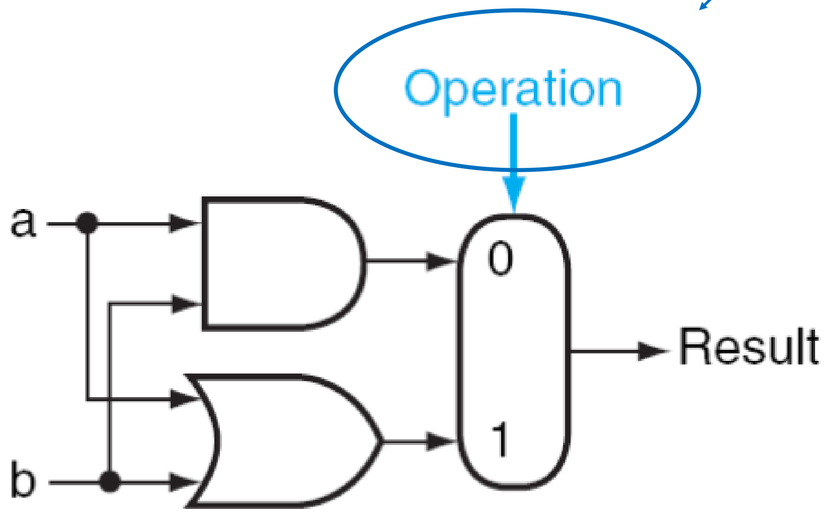
d	c
0	a
1	b

UNIVERSITÀ DEGLI STUDI DI TRIESTE

ALU A 1 BIT: OPERAZIONI LOGICHE

ALU su 1 bit che implementa AND e OR

È il segnale di controllo del mux e sceglie QUALE OPERAZIONE svolgere

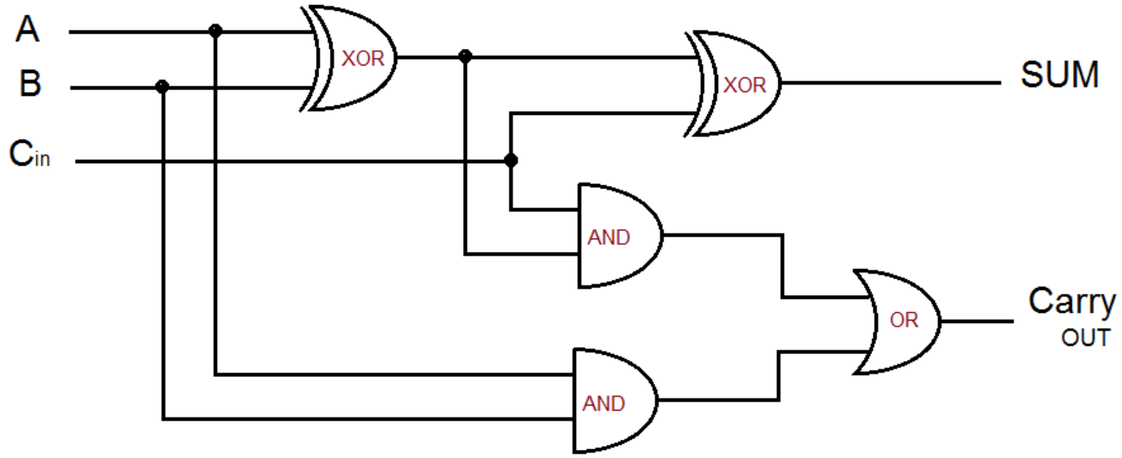


ALU A 1 BIT: OPERAZIONI ARITMETICHE

Addizione (carry, sum)

Inputs			Outputs	
A	B	C_{in}	S	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

Full adder

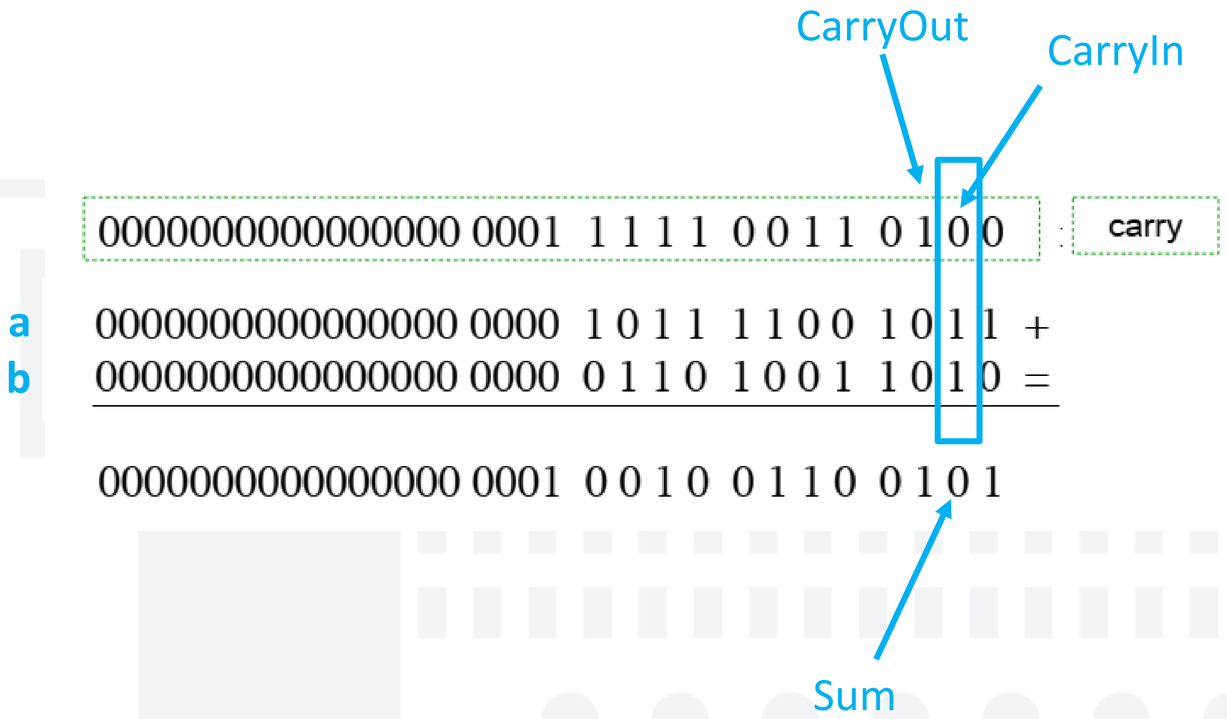


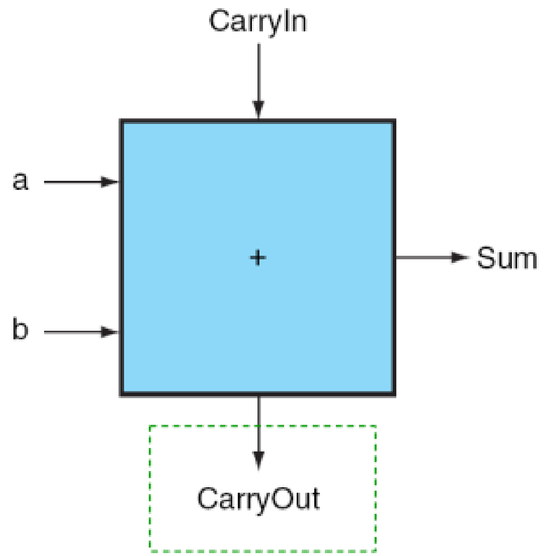
S = (A $\oplus$ B) $\oplus$ C

$\mathbf{C_{out}}$ = (A $\oplus$ B) · C + A · B

ALU A 1 BIT: OPERAZIONI ARITMETICHE

Addizione

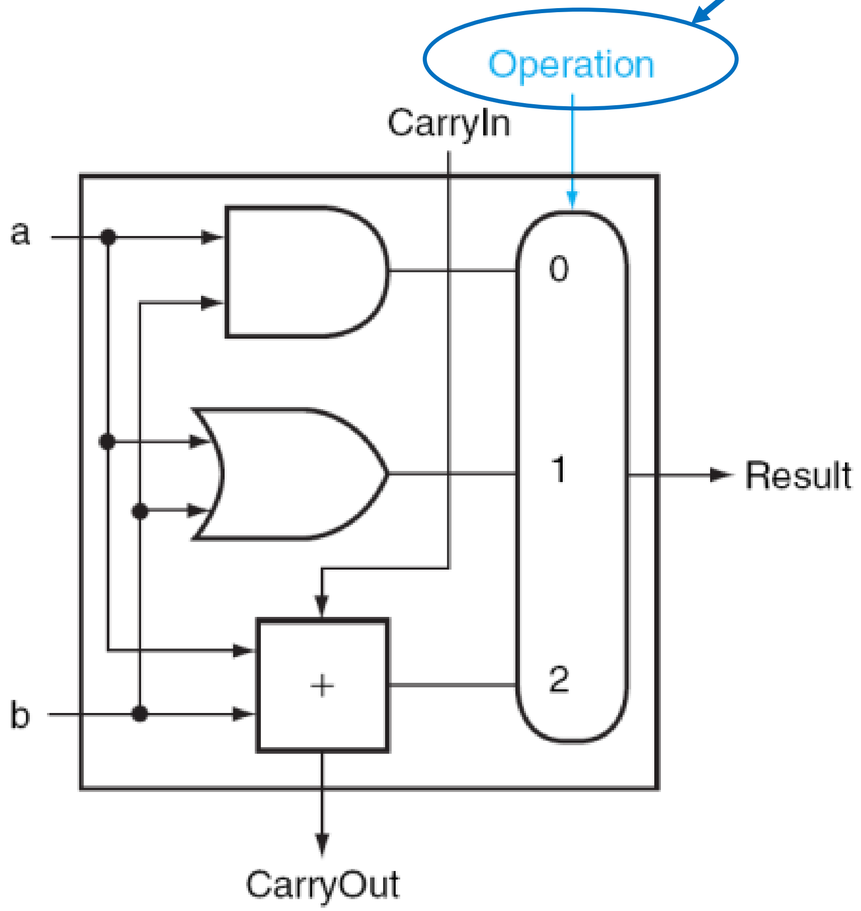




ALU A 1 BIT: SCELTA DI DIVERSE OPERAZIONI

ALU a 1 bit che può eseguire SOMMA, AND oppure OR

Non basta più 1 bit, **servono 2 bit** per scegliere UNA tra tre operazioni possibili

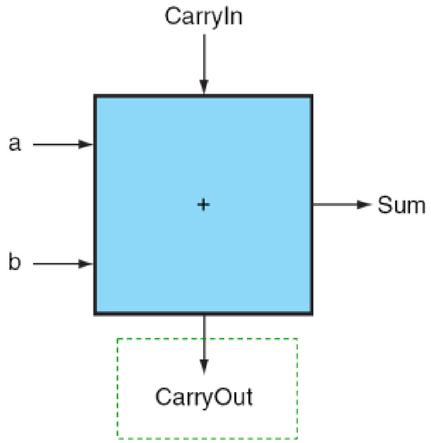


UNIVERSITÀ DEGLI STUDI DI TRIESTE

ALU A 1 BIT: POSSIAMO EFFETTUARE UNA SOTTRAZIONE?

Cosa succede se settiamo CarryIn a 1?

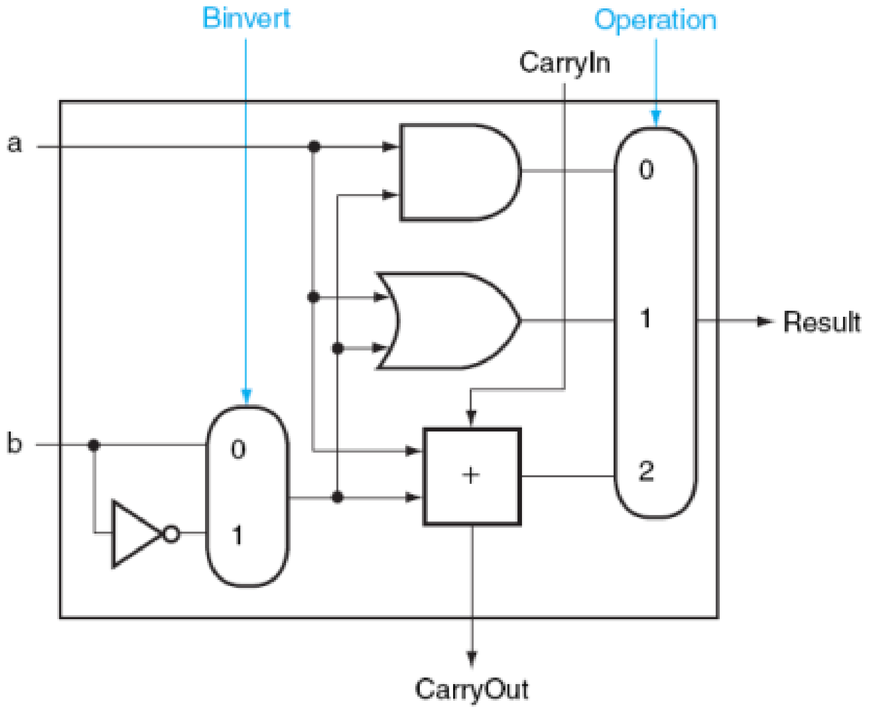
Vuol dire sommare a+b+1



Se neghiamo b, possiamo ottenere una sottrazione (in CA2)

$$a - b = a + (\bar{b} + 1)$$

Invertire tutti i bit, uno ad uno



UNIVERSITÀ DEGLI STUDI DI TRIESTE

ALU A 1 BIT

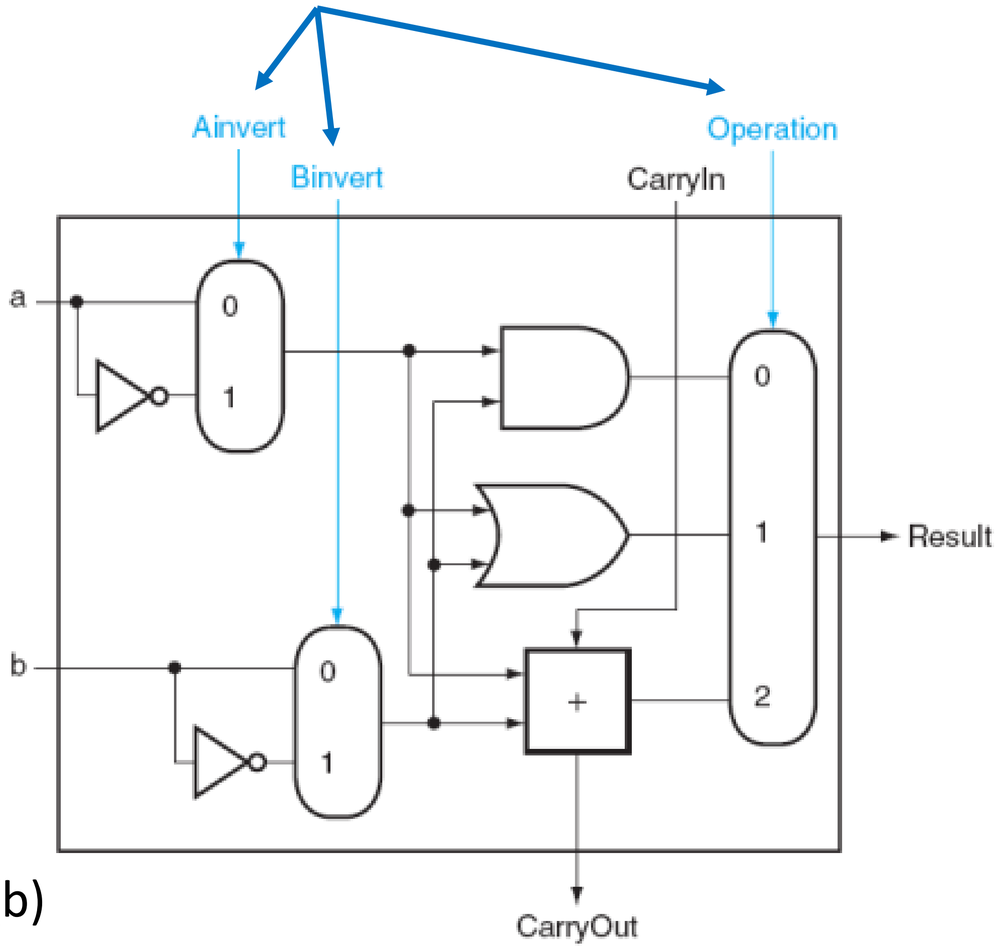
NOTA: Non aumentiamo il numero di bit del segnale di controllo Operation, ma aggiungiamo due segnali di controllo da 1 bit ciascuno (Ainvert, Binvert).

Come si implementano anche il NOR e NAND?

Leggi di De Morgan

$$\overline{a + b} = \bar{a} \cdot \bar{b}$$

$$\overline{a \cdot b} = \bar{a} + \bar{b}$$



NOR(a,b) = NOT(a+b) = NOT(a) · NOT(b) = NOT(a) **AND** NOT(b)

NAND(a,b) = NOT(a · b) = NOT(a) + NOT (b) = NOT(a) **OR** NOT(b)

E se vogliamo operare su più bit

contemporaneamente, per esempio su 32 bit??

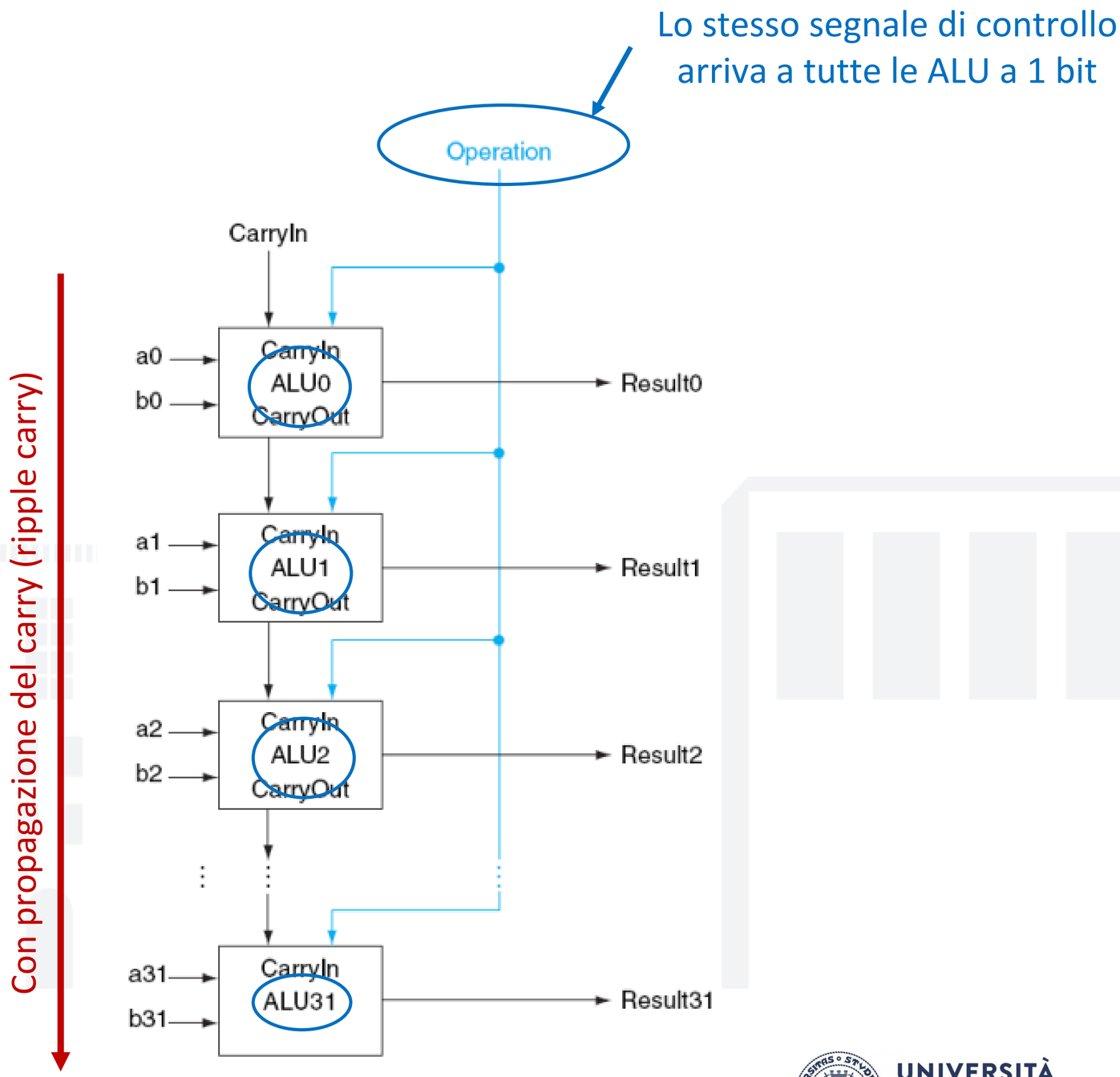
$$\begin{array}{l} 00000000000000000000\ 1011\ 1100\ 1011\ + \\ 00000000000000000000\ 0110\ 1001\ 1010\ = \\ \hline \end{array}$$

Per esempio, vogliamo calcolare la somma tra questi due numeri a 32 bit. Come si fa?

ALU A **32 BIT**

Connessione di 1-bit ALU adiacenti

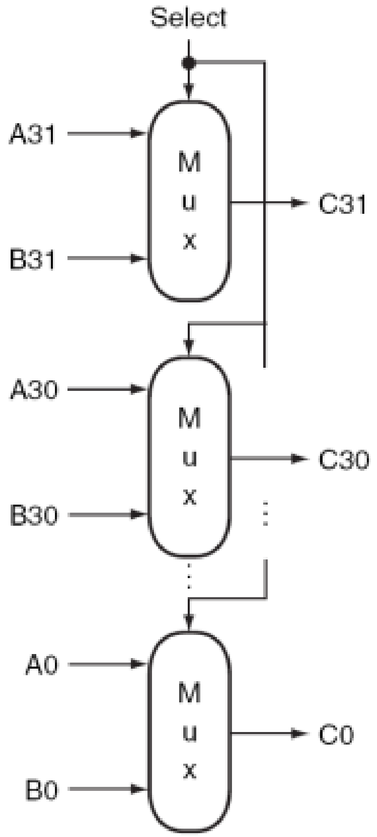
RIPPLE CARRY

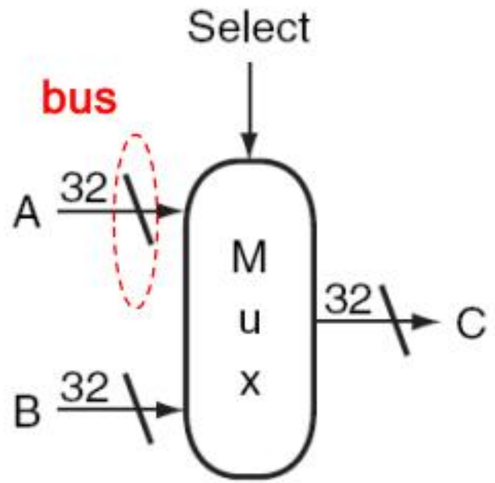


UNIVERSITÀ DEGLI STUDI DI TRIESTE

ARRAY DI ELEMENTI LOGICI

- La maggior parte delle operazioni vengono svolte su 32 bit: necessità di creare array di elementi logici.
- Un bus è una collezione di linee di input che verranno trattate come un singolo segnale.
- Un multiplexor con un bus a 32-bit corrisponde ad un array di 32 multiplexor ad 1-bit.





UNIVERSITÀ DEGLI STUDI DI TRIESTE

ALU A 1 BIT: SET ON LESS THEN (SLT)

slt registro1, registro2, registro3

Poni registro1=1 ***se*** il valore contenuto (a) in registro2 è minore del valore contenuto (b) in registro3.

Altrimenti, se il valore contenuto in registro2 non è minore del valore contenuto in registro3, allora poni registro1=0.

Risultato: 1 se a<b, 0 altrimenti.

Va calcolata la differenza su 32 bit tra il numero a e il numero b

ALU A 1 BIT: SET ON LESS THEN (SLT)

Risultato: 1 se a<b, 0 altrimenti. L'output ha 32 bit ma tutti i bit dal bit-1 al bit-31 saranno a zero (quello di valore minimo tra i 32) e **solo il bit-0 verrà aggiornato** con il valore dell'operazione di slt.

a<b ⇔ a-b<0
SE la sottrazione (a - b) è negativa
ALLORA bit-31 della sottrazione (a - b)=1

(bit-31 è quello che porta il segno)

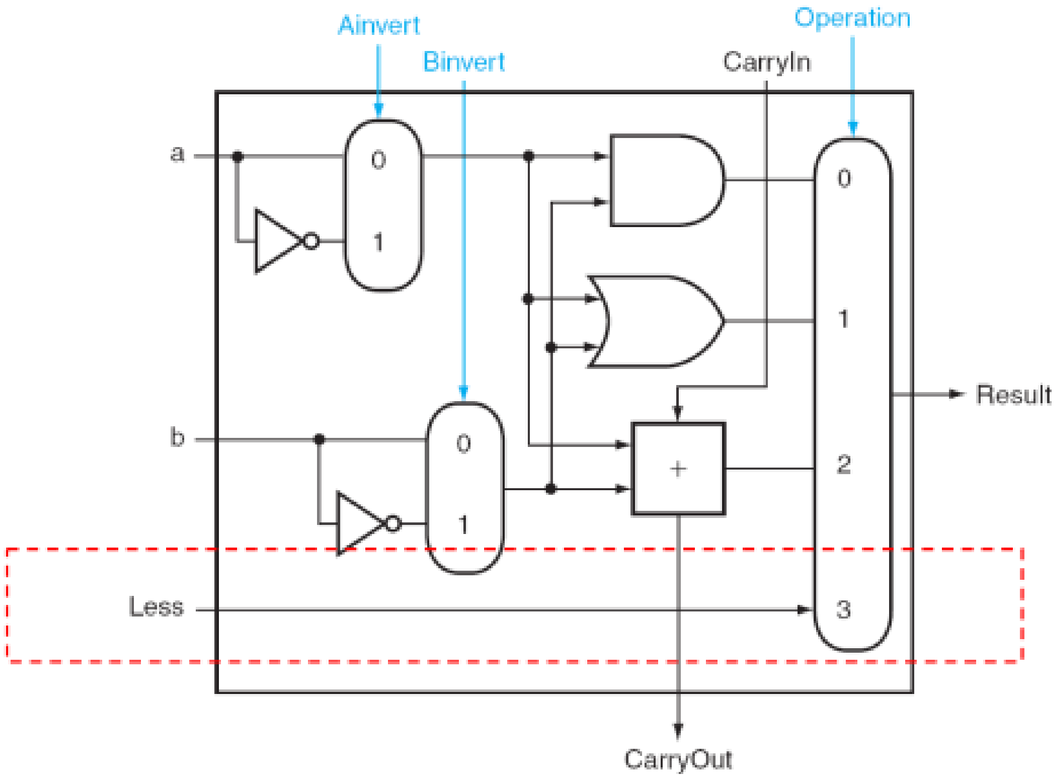
Per poter realizzare il confronto si effettua la sottrazione tra a e b su 32 bit

- Se a-b è minore di 0, allora a<b (per l'istruzione slt)
  - ➢ il risultato deve essere **00...01**
- Se a-b è maggiore di 0, allora a>b (per l'istruzione slt)
  - ➢ il risultato deve essere **00...00**

Nota. Il risultato dell'istruzione di slt è **1 bit** (quindi il bit meno significativo e tutti gli altri 31 possono essere azzerati).

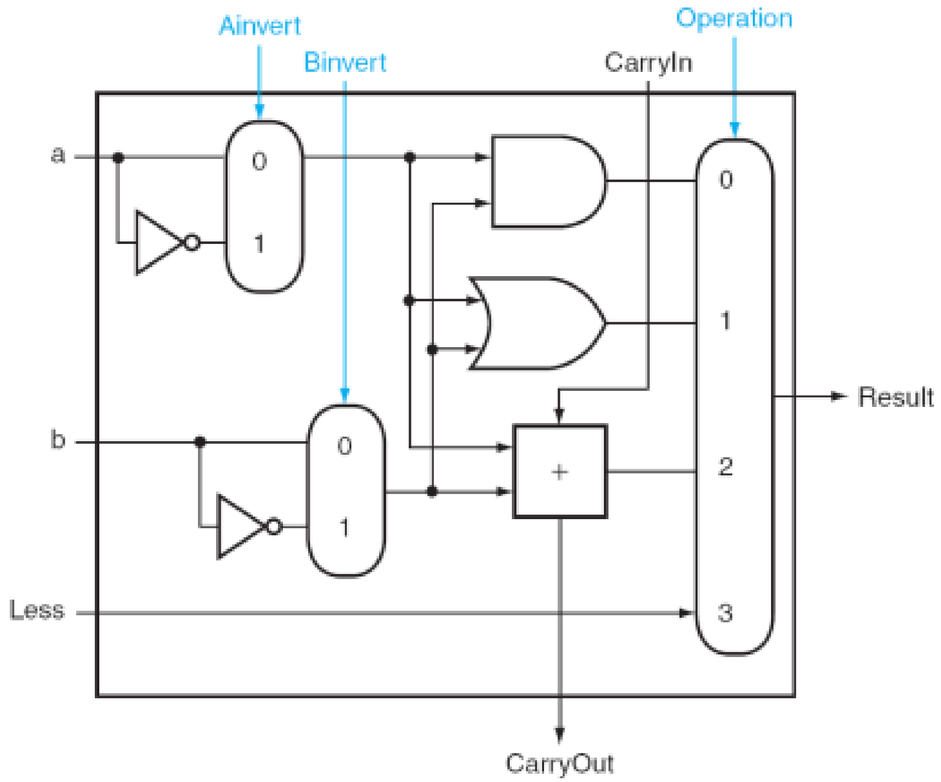
ALU A 1 BIT: SLT

ALU a 1 bit: aggiungiamo opzione per realizzare anche SLT



ALU A 32 BIT: **SLT**

ALU0 – ALU30



ALU0:

CarryIn = 1 (per fare la sottrazione)

Less = … (v. slide successiva)

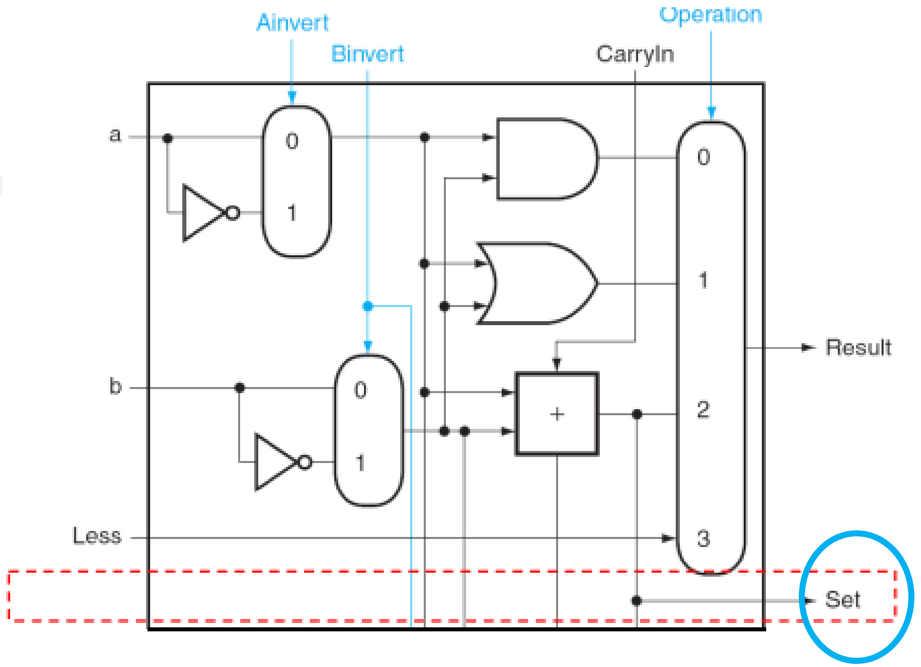
ALU1-ALU30:

$CarryIn_i = CarryOut_{i-1}$

Less = 0

ALU A 32 BIT: SLT

ALU31



ALU31:

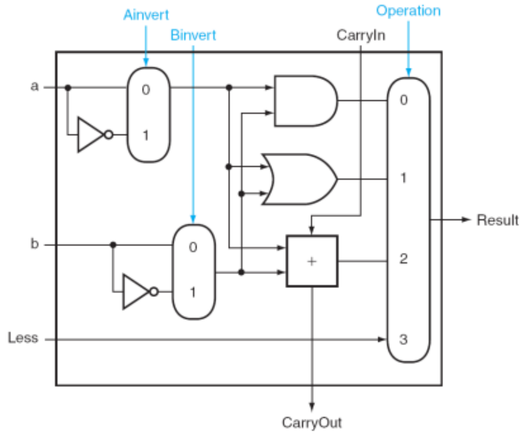
$CarryIn_{31} = CarryOut_{30}$

Less = 0

Set = bit-31 (segno)

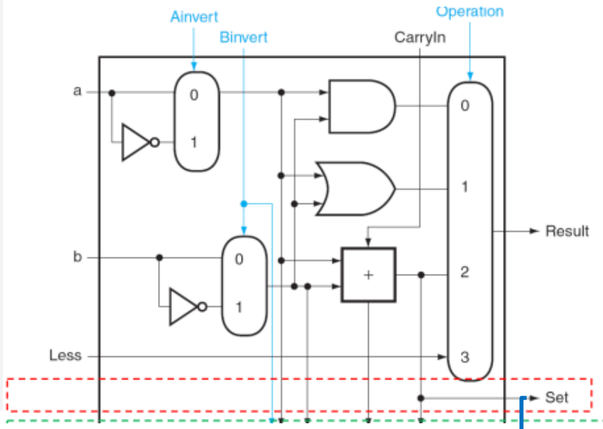
ALU A 32 BIT: SLT

ALU0



:

ALU31



ALU0:

CarryIn = 1 (per fare la sottrazione)

Less = Set

TEST DI OVERFLOW

Per assicurarsi della correttezza del risultato di un'operazione di sottrazione in CA2 bisogna verificare l'assenza di overflow:

- *non* si ha overflow se gli operandi hanno segno concorde
- si ha overflow *se gli operandi hanno segno discorde* e il segno del risultato è discorde con il segno del primo operando

Casi di overflow:

- (A - B)> 0 e bit-31 di (A - B)=1 oppure
- (A - B)< 0 e bit-31 di (A - B)=0

bit-31(A) = 0, bit-31(B) = 1 bit-31(A - B) = 1

oppure

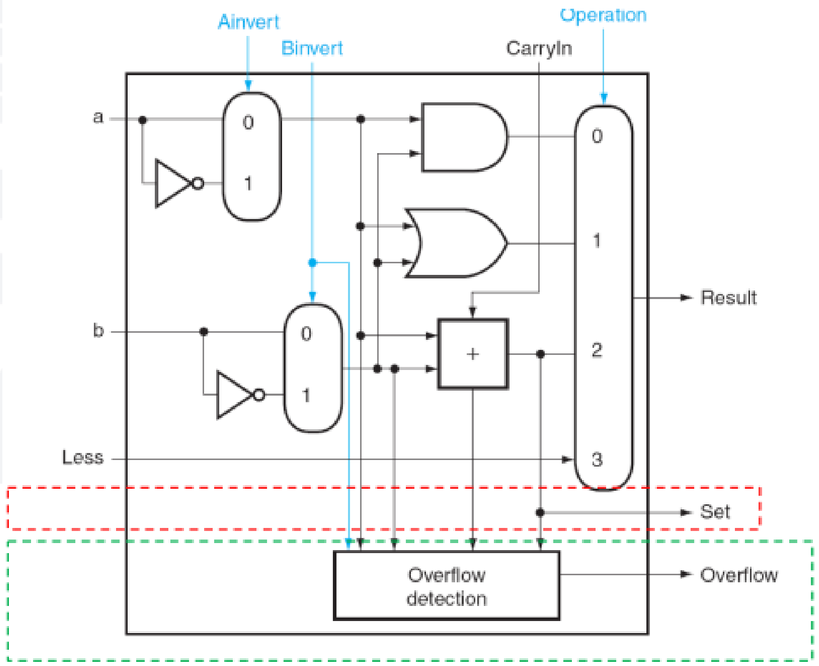
bit-31(A) = 1, bit-31(B) = 0 bit-31(A - B) = 0

A	B	A-B	OVERFLOW
0	1	1	1
1	0	0	1

Controllo di overflow:

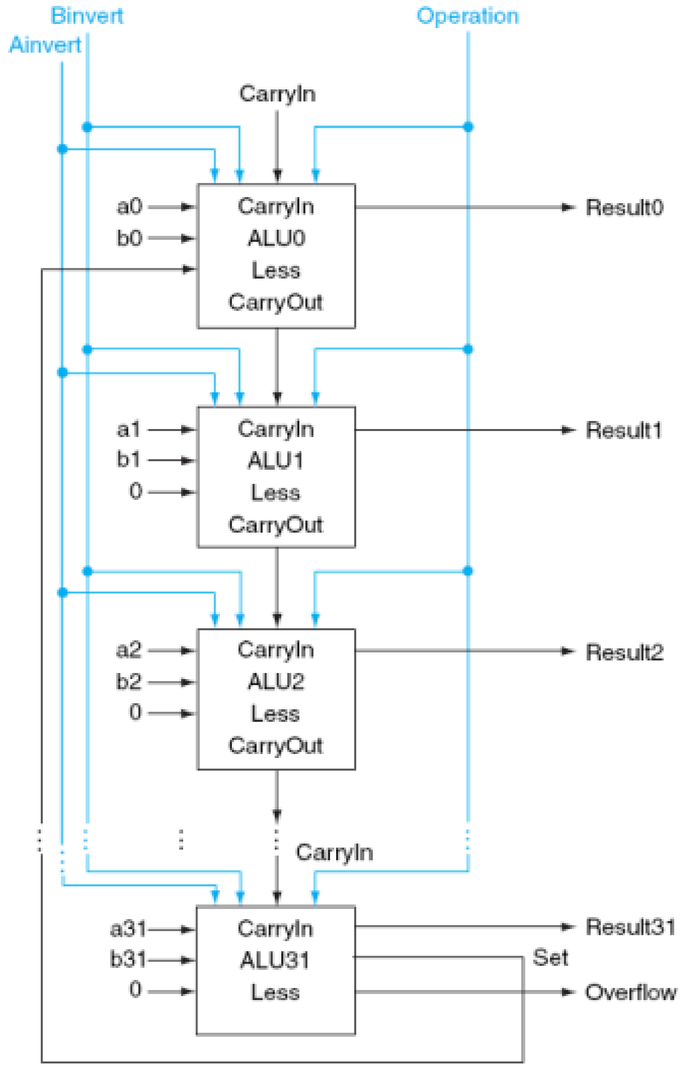
$$overflow = \bar{a} \cdot b \cdot Result + a \cdot \bar{b} \cdot \overline{Result}$$

$$\text{set} = \text{bit-31(Result)} \oplus \text{overflow}$$



UNIVERSITÀ DEGLI STUDI DI TRIESTE

ALU A 32 BIT CON SLT + OVERFLOW



- **Per sottrazioni**: Binvert e CarryIn = 1
- **Per addizioni e operazioni logiche**: Binvert e CarryIn = 0

Potrebbe servire ancora qualcosa?

Se volessimo realizzare una branch on equal (beq)?

OUTPUT AGGIUNTIVO PER LA BRANCH ON EQUAL (BEQ)

beq registro1, registro2, L1

Vai all'istruzione etichettata con L1 se il valore contenuto in registro1 è uguale al valore contenuto in registro2

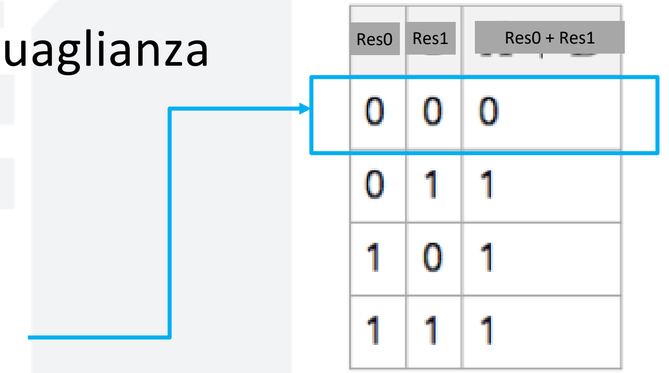
Per verificare l'uguaglianza di a e b: sottrazione

- a-b = 0 ↔ a = b

- **Se a=b** allora dobbiamo mettere a 1 la linea di output usata per l'uguaglianza

Soluzione: OR di tutte le uscite e poi negare il risultato

$$Zero = \overline{(Result31 + Result30 + \cdots + Result0)}$$

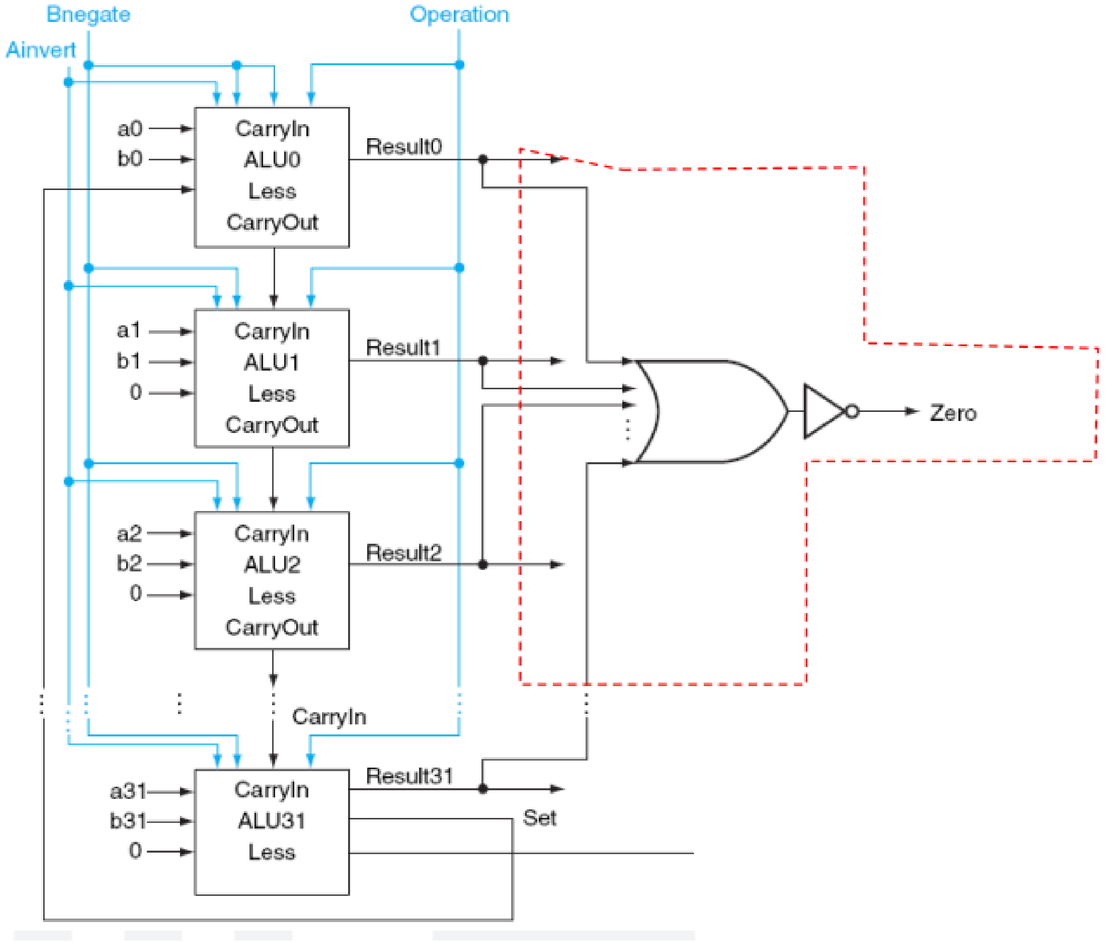


OUTPUT AGGIUNTIVO PER LA BRANCH ON EQUAL (BEQ)

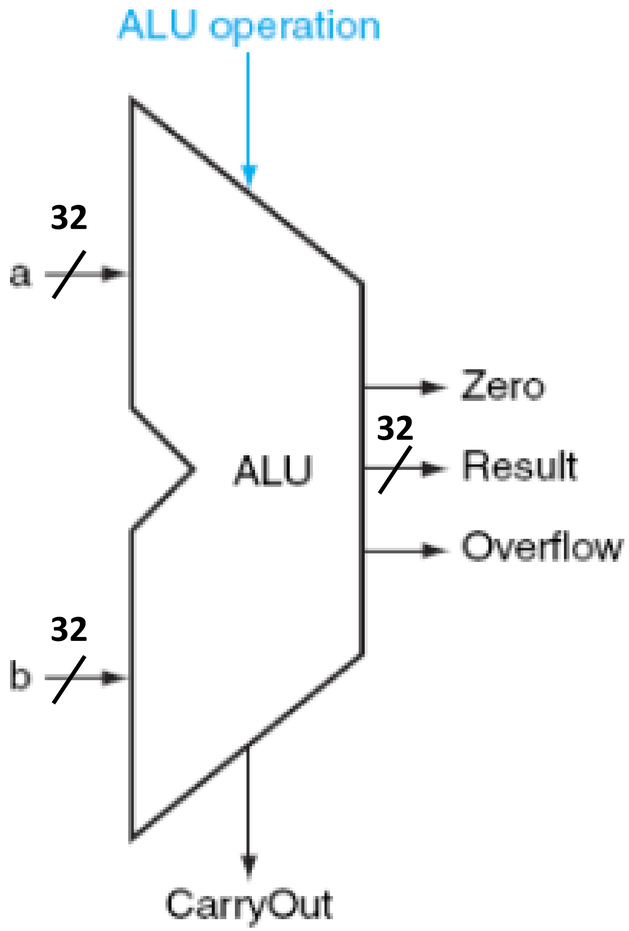
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

4 linee di controllo:
1 bit – Ainvert
1 bit – Binvert
2 bit – 4 possibili operazioni

Esempio: «subtract» → 0110 → Ainvert=0, Binvert=1 (ovvero, abbiamo bisogno di not(b) e CarryIn=1), $10_2 = 2$ attiva la linea del MUX per la somma/sottrazione.



ALU: RAPPRESENTAZIONE SIMBOLICA



Realizzata così, questa ALU supporta le seguenti operazioni logico-artimetiche:

add $t0, $t1, $t2

and $t0, $t1, $t2

or $t0, $t1, $t2

nor $t0, $t1, $t2

slt $t0, $t1, $t2

:

Programma per il I parziale

- Rappresentazione dell’informazione, assembly, ISA, catena programmativa, procedure annidate, stack, QtSpim, algebra booleana, funzioni logiche, circuiti in logica combinatoria (tutto il programma svolto fino al giorno 18/03/2026 (compreso)

- Escluso circuiti in logica sequenziale

- Esempio di esercizi: prossima slide

Esempi

Qual è l'intervallo dei valori rappresentabili in complemento a due su 5 bit?

- ☐ -15 … +16
- ☐ -15 … +15
- ☐ 0 … +31
- ☐ -16 … +15
- ☐ -16 … +16
- ☐ nessuna delle precedenti

Convertire il numero $(1011{,}011)_2$ in decimale.

Data la seguente tabella di verità, disegnare il circuito corrispondente. Utilizzare l'algebra booleana per ottenere l'espressione di F ed S in funzione di A, B, C (riportare tutti i passaggi). Gli input sono: A, B, C. Gli output sono: F, S. Nel disegnare il circuito, usare le porte logiche con le convenzioni di rappresentazioni note. Aggiungere, all'interno delle stesse, il nome della funzione che svolgono (OR, AND, ..).

A	B	C	F	S
0	0	0	**1**	**0**
0	0	1	**1**	**1**
0	1	0	**0**	**1**
0	1	1	**0**	**0**
1	0	0	**0**	**0**
1	0	1	**0**	**1**
1	1	0	**1**	**1**
1	1	1	**1**	**1**

4. Convertire la seguente istruzione dal formato binario al formato mnemonico assembly:
`0000 0001 1000 1101 0001 0000 0010 0010`

- ☐ `add $t0, $a0, $s5`
- ☐ nessuna delle altre risposte è corretta
- ☐ `sub $v0, $t5, $t5`
- ☐ non saprei
- ☐ `add $v0, $t4, $t5`
- ☐ `sub $t0, $a0, $s5`

Dato il seguente segmento di codice assembly, quale valore assume il registro `$v0` alla fine della esecuzione?

```
.text
main:
li $t0, 21
li $s2, 0x00f00f0f
and $t5, $t0, $s2
sll $v0, $t5, 3
# fine esecuzione
```

- ☐ 10
- ☐ 20
- ☐ 5
- ☐ 80
- ☐ nessuna delle altre risposte è corretta
- ☐ 40

Semplificare la seguente funzione logica usando l'algebra booleana (specificare, dove usati, assiomi/proprietà/teoremi):

$$F(A,B,C) = A \cdot B + A \cdot \overline{B} + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{\overline{B} + C}$$

Materiale per la lezione

- Patterson & Hennessy, Appendice B
- Patterson & Hennessy, §4.4 (controllo ALU)

Prossima lezione: 25 ***marzo, h.14:00, aula*** 3B