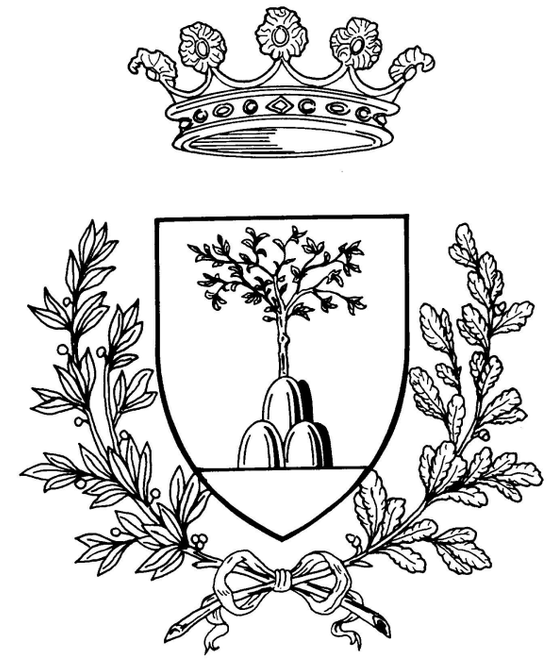


Architettura degli Elaboratori e Laboratorio

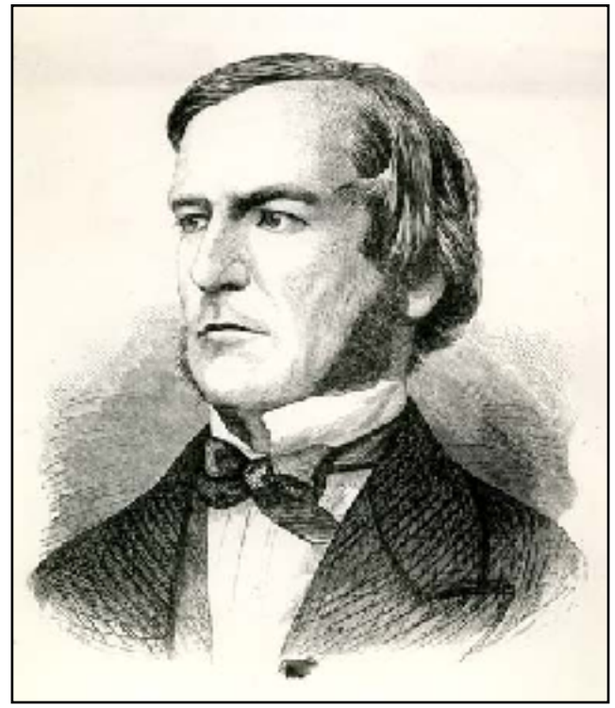
Matteo Manzali

Università degli Studi di Ferrara

Anno Accademico 2016 - 2017

Algebra booleana

- L’algebra booleana è un particolare tipo di algebra in cui le variabili e le funzioni possono solo avere valori 0 e 1.
- Deriva il suo nome dal matematico inglese George Boole che la ideò.
- Le funzioni dell’algebra booleana sono isomorfe ai circuiti digitali:
  - un circuito digitale può essere espresso tramite un’espressione booleana e viceversa.
- Una funzione booleana ha una o più variabili in input e fornisce risultati che **dipendono solo** da queste variabili.



Algebra booleana

- Le variabili possono assumere solo i valori 0 o 1:
  - una funzione booleana con **n** variabili di input ha solo **2^n** combinazioni possibili
  - può essere descritta attraverso una tabella, detta **tabella di verità**, con 2^n righe

A	B	C	V
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Esempio:

- V = f (A, B, C)
- 3 variabili (A, B, C)
- 2^3 → 8 righe
- A, B, C → input
- V → output

Operatori di base

- L'algebra booleana si basa su tre operatori di base:
  - **OR** → "somma logica": A + B = 1 se almeno uno tra A e B è uguale a 1
  - **AND** → "prodotto logico": A • B = 1 se entrambi A e B sono uguali a 1
  - **NOT** → operazione di "complemento": $\overline{A} = 1$ se A = 0 e $\overline{A} = 0$ se A = 1
- Con le porte logiche AND, OR e NOT è possibile realizzare qualsiasi funzione booleana.
- Ogni operatore ha associata una astrazione di un dispositivo elettronico che lo implementa.

Operatore OR

- Tabella di verità:

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

- Astrazione elettronica:

Operatore AND

- Tabella di verità:

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

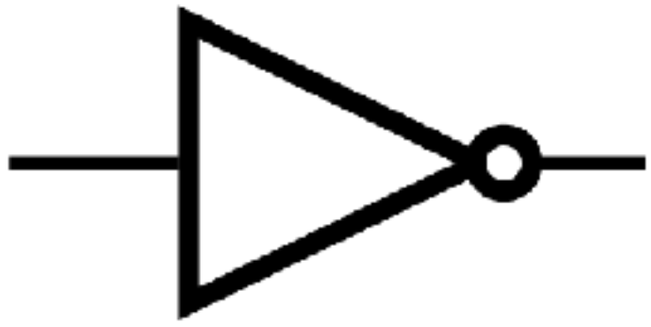
- Astrazione elettronica:

Operatore NOT

- Tabella di verità:

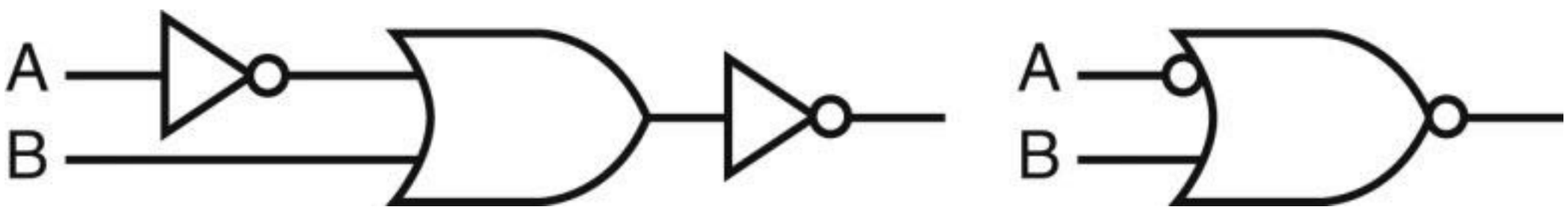
A	$\overline{A}$
0	1
1	0

- Astrazione elettronica:



- Versione semplificata dell'astrazione elettronica del NOT:

Es.: $\overline{\overline{A} + B}$



Altri operatori

- Esistono altri operatori nell'algebra booleana:
  - NOR
  - NAND
  - XOR
  - XNOR
- **Tutti questi operatori possono essere riprodotti attraverso l'uso degli operatori di base (OR, AND, NOT)**.

Operatore NOR

- Rappresenta la negazione del risultato dell'operatore OR.
- Tabella di verità:

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

- Astrazione elettronica:

Operatore NAND

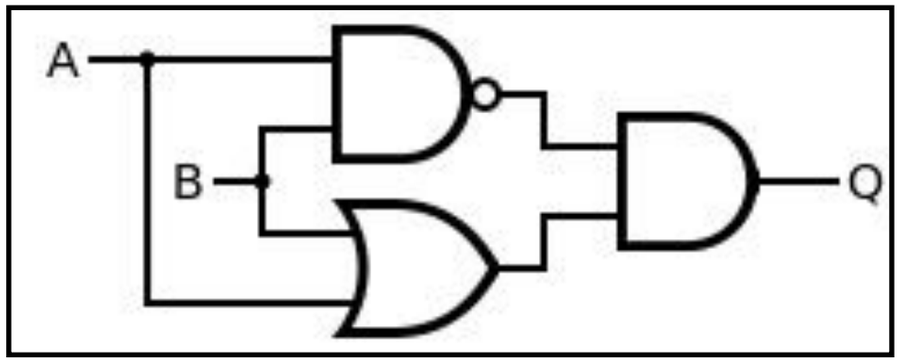
- Rappresenta la negazione del risultato dell'operatore AND.
- Tabella di verità:

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

- Astrazione elettronica:

Operatore XOR

- Restituisce 1 se e solo se il numero degli operandi uguali a 1 è **dispari**:
  - somma modulo 2 (senza riporto)
  - bit di parità



- Astrazione elettrica:

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Operatore XNOR

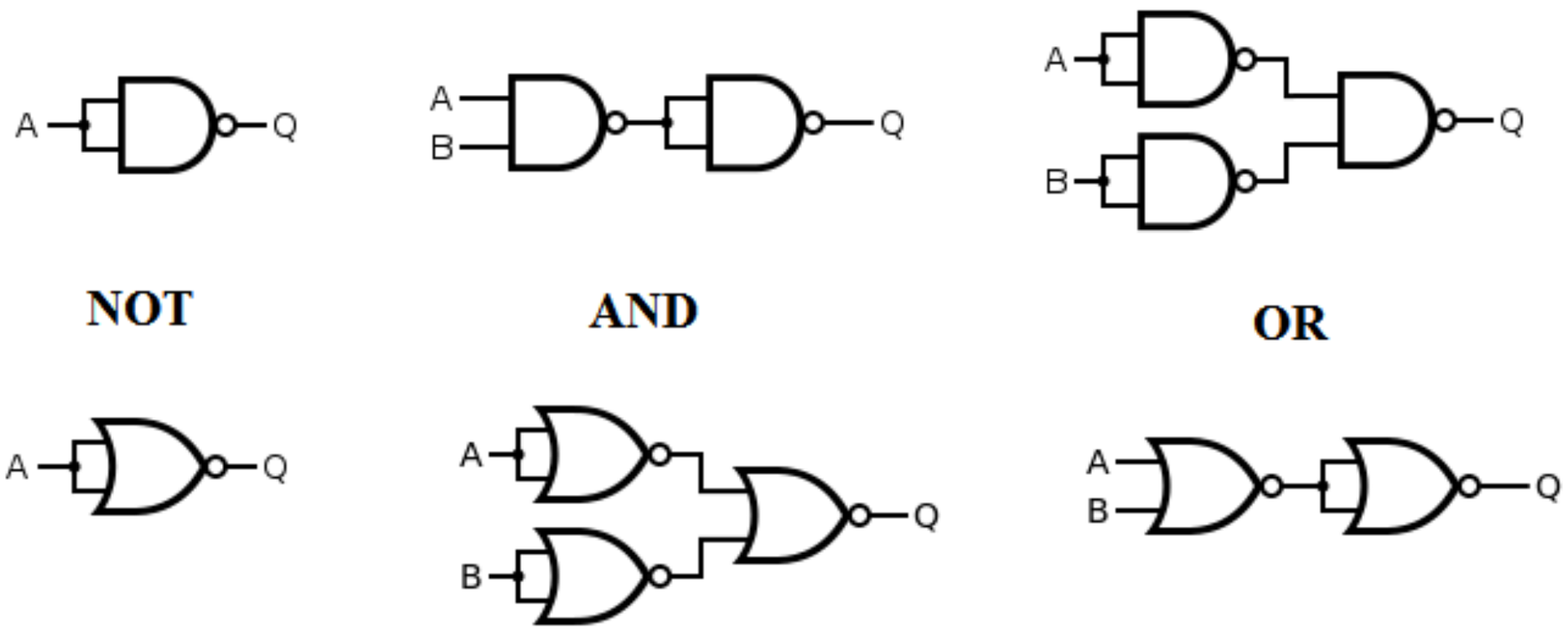
- Rappresenta la negazione del risultato dell'operatore XOR.
- Tabella di verità:

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

- Astrazione elettronica:

Operatori universali

- NAND e NOR sono definiti **operatori universali**:
  - è possibile implementare gli operatori di base (OR, AND, NOT) in funzione di soli NAND o soli NOR
  - utilizzati nell'implementazione di circuiti digitali per via della riduzione di costi e spazi



Proprietà algebra booleana

- Identità:

$$A + 0 = A \quad e \quad A \cdot 1 = A$$

- Assorbimento:

$$A + 1 = 1 \quad e \quad A \cdot 0 = 0$$

- Inverso:

$$A + \overline{A} = 1 \quad e \quad A \cdot \overline{A} = 0$$

- Idempotenza:

$$A + A = A \quad e \quad A \cdot A = A$$

Proprietà algebra booleana

- Commutativa:

  $A + B = B + A \quad e \quad A \cdot B = B \cdot A$

- Associativa:

  $A + (B + C) = (A + B) + C \quad e \quad A \cdot (B \cdot C) = (A \cdot B) \cdot C$

- Distributiva:

  $A \cdot (B + C) = (A \cdot B) + (A \cdot C) \quad e \quad A + (B \cdot C) = (A + B) \cdot (A + C)$

- Assorbimento:

  $A + \overline{A} \cdot B = A + B$

Teoremi di De Morgan

- I teoremi (o leggi) di De Morgan stabiliscono relazioni di equivalenza tra gli operatori di congiunzione logica AND e OR.
- Sono utilizzati per l'analisi e la semplificazione di circuiti digitali.
- I teoremi sono due:
  1. $\overline{A \cdot B} = \overline{A} + \overline{B}$
  2. $\overline{A + B} = \overline{A} \cdot \overline{B}$
- I teoremi di De Morgan si possono estendere a più di due variabili (A, B, C, D, …):
  - $\overline{A \cdot B \cdot C \cdot D} = \overline{A} + \overline{B} + \overline{C} + \overline{D}$

Funzioni e tabelle di verità

Parametri in ingresso

A	B	C	T	U	V
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	1	0

Funzioni di uscita

- $T = A + B + C$
- $U = A \cdot B \cdot C$
- $V = ((A \cdot B) + (A \cdot C) + (B \cdot C)) \cdot \overline{(A \cdot B \cdot C)}$

Forme canoniche

- La forma canonica (o forma normale) di una funzione booleana è una rappresentazione di un'espressione booleana ricavabile dalla tabella di verità.
- Esistono due forme canoniche:
  - **“somma di prodotti”** (forma disgiuntiva)
  - **“prodotto di somme”** (forma congiuntiva)

Somma di prodotti

- Per ogni valore della funzione pari a 1:
  - ne prendiamo i parametri in ingresso
  - se il valore di un parametro è 0 lo neghiamo
  - facciamo il prodotto dei parametri (mintermine)
- Sommiamo i prodotti ottenuti.

Somma di prodotti

A	B	C	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

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

mintermine

$$V = A\overline{B}C + AB\overline{C} + ABC$$

il • è implicito ed ha precedenza sul +

Prodotto di somme

- Per ogni valore della funzione pari a 0:
  - ne prendiamo i parametri in ingresso
  - se il valore di un parametro è 1 lo neghiamo
  - facciamo la somma dei parametri (maxtermine)
- Facciamo il prodotto delle somme ottenute.

Prodotto di somme

A	B	C	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

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

maxtermine

Forme canoniche

- Quale forma canonica utilizzare?
  - non c'è una regola
  - di solito la somma di prodotti è la più intuitiva
  - dipende anche dalla quantità di 1 e di 0 nella funzione
- Una volta scritta la funzione in forma canonica, la si può ottimizzare utilizzando le proprietà dell'algebra booleana.
- Questo permette di ridurre la quantità di operazioni previste dalla funzione:
  - risparmio in termini di costi, spazio e consumo nell'implementazione di circuiti elettrici

Esempio

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

La funzione “Y” è espressa in prima forma canonica.

Esempio

$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

Sfruttiamo la proprietà dell'idempotenza e aggiungiamo 3 mintermini ABC (mintermine già esistente):

$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC + ABC + ABC + ABC$

Utilizziamo la proprietà distributiva:

$Y = BC(A + \overline{A}) + AC(B + \overline{B}) + AB(C + \overline{C})$

Utilizziamo la proprietà dell'inverso ($A + \overline{A} = 1$):

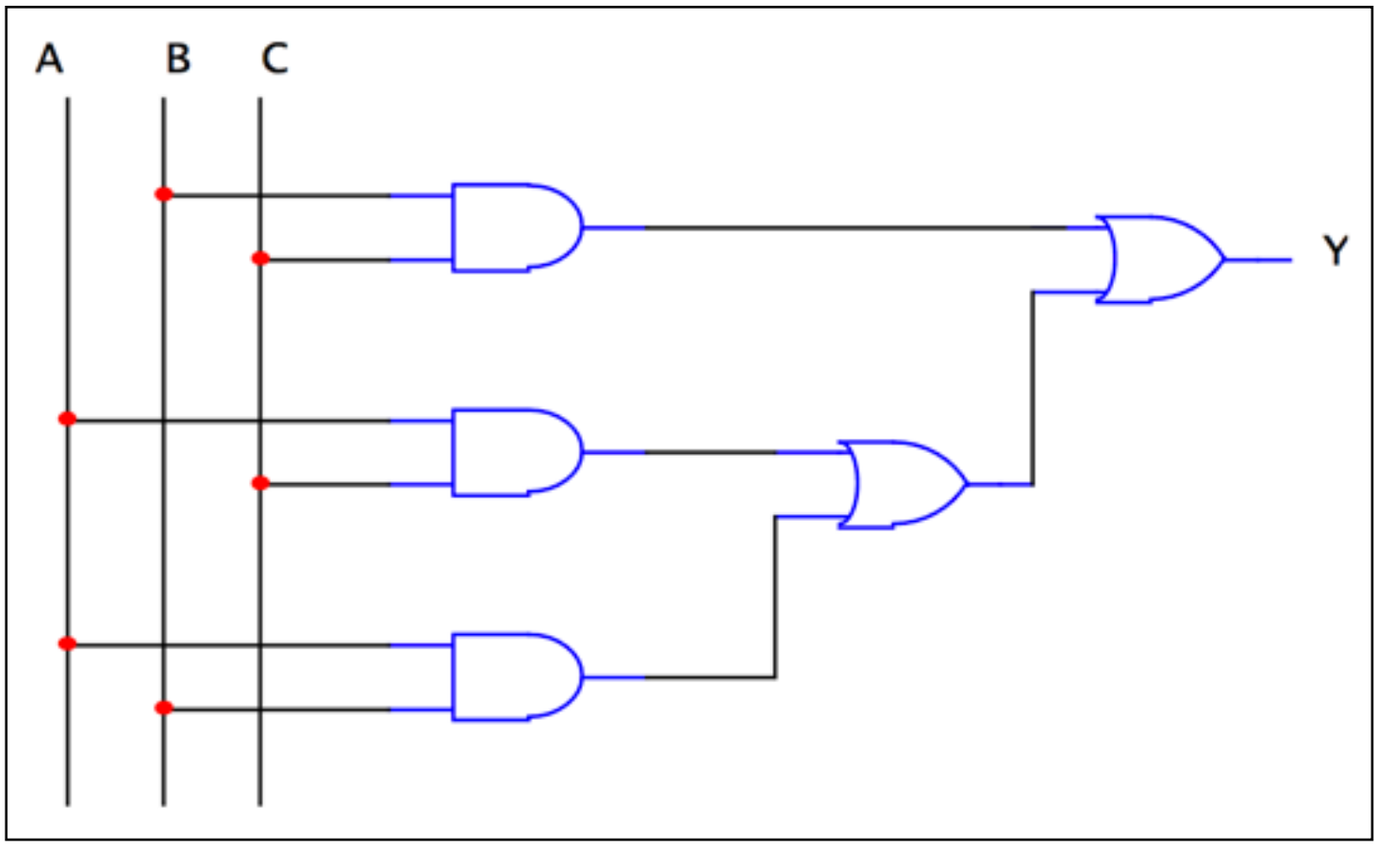
$Y = BC1 + AC1 + AB1$

Utilizziamo la proprietà dell'identità ($A \cdot 1 = A$):

$Y = BC + AC + AB$

Esempio

- L'espressione così minimizzata è $Y = BC + AC + AB$
- Il circuito logico sarà costituito da:
  - tre porte AND
  - due porte OR (o una porta OR a tre ingressi)

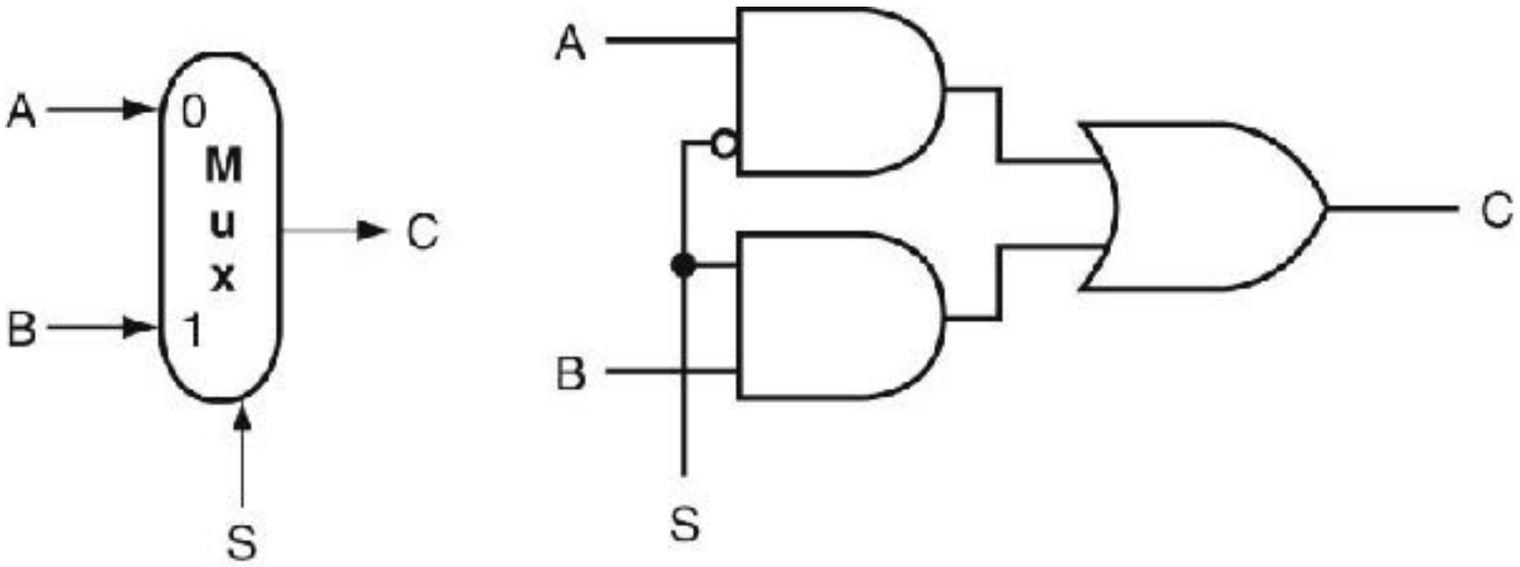


ALU

- La ALU (Arithmetic and Logic Unit) o unità aritmetico-logica è il componente hardware che svolge le operazioni aritmetiche e logiche.
- L'ALU è una componente fondamentale della CPU.
- Nelle prossime slides vedremo una ALU semplificata in grado di eseguire le seguenti operazioni su interi signed a 32 bit:
  - somma e sottrazione
  - AND
  - OR
  - altro?
- **E' la ALU dell'architettura MIPS.**

Multiplexer

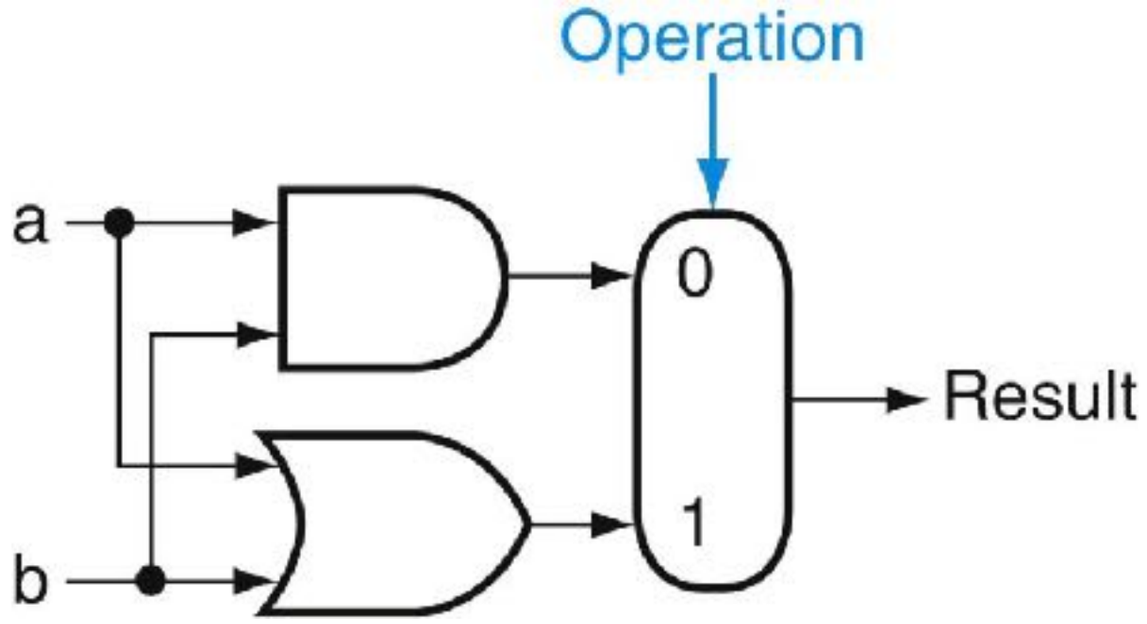
- Funzione logica che permette di replicare in output una delle variabili in input in base al valore di una o più variabili di controllo:



- S è il bit di controllo:
  - S = 0 → C = A e S = 1 → C = B
- **Con N bit di controllo posso gestire 2^N bit in input.**

ALU - operazioni logiche

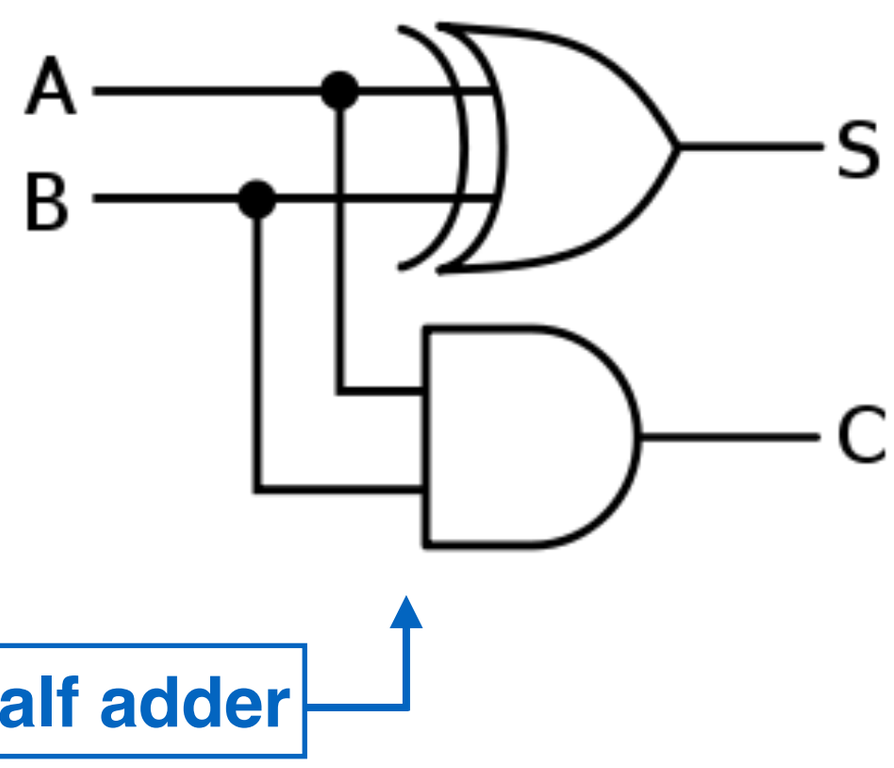
- Le operazioni AND e OR nella ALU sono svolte dai corrispondenti circuiti elettrici.
- Un multiplexer sceglie quale delle due operazioni deve essere svolta.



ALU - Half adder

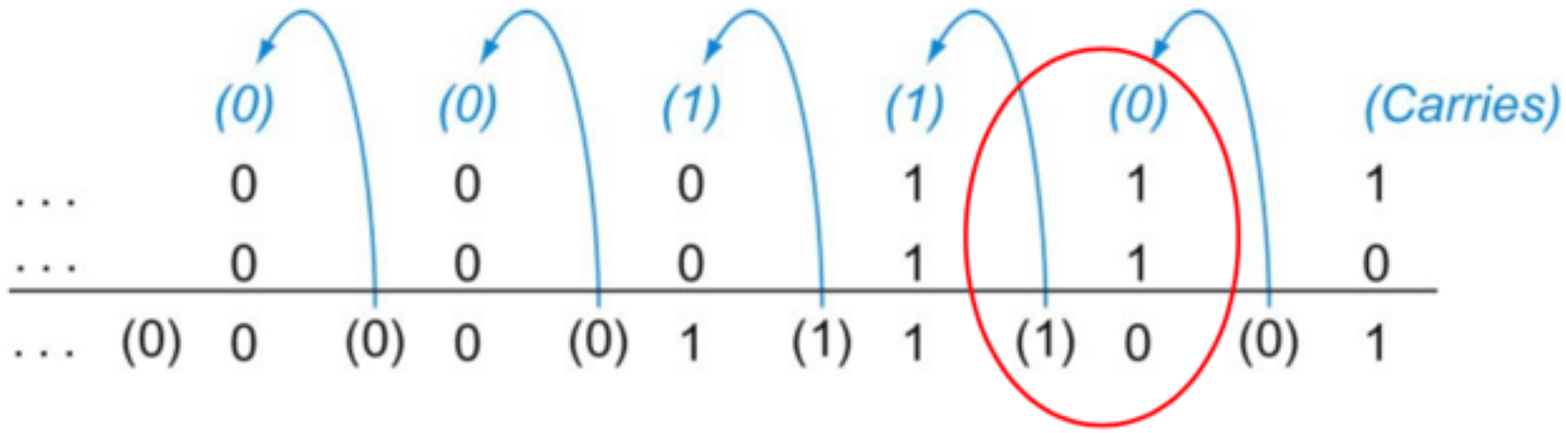
- La funzione somma tra due bit genera due bit in output:
  - la somma (S) → vale 1 se solo uno dei due bit vale 1 (XOR)
  - il riporto (C) → vale 1 se entrambi i bit valgono 1 (AND)

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



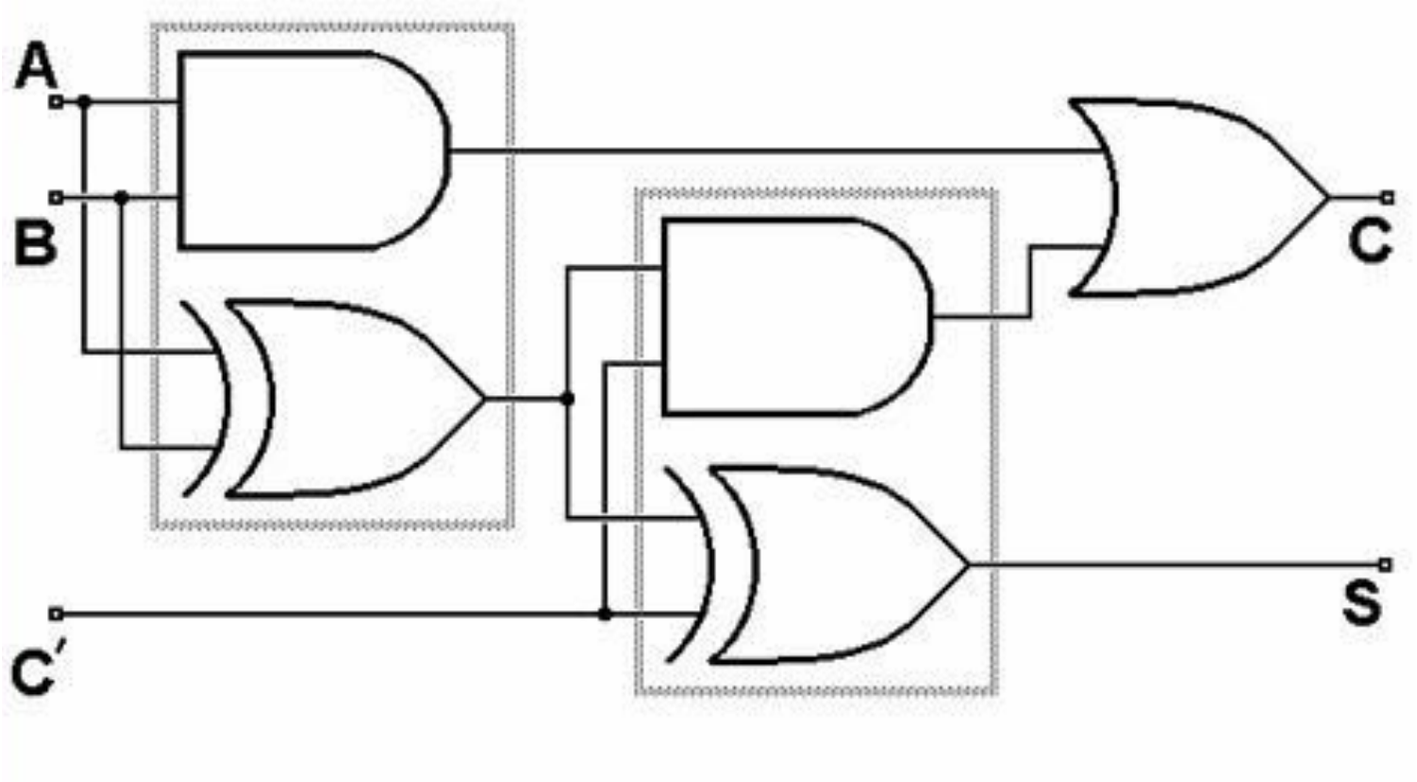
ALU - Carry In

- L’addizione di numeri composti da più di 1 bit comporta un problema:
  - potrebbe esserci un riporto “pendente” dall’addizione dei bit nella posizione precedente
  - questo riporto diventa un terzo input della addizione (Carry In)



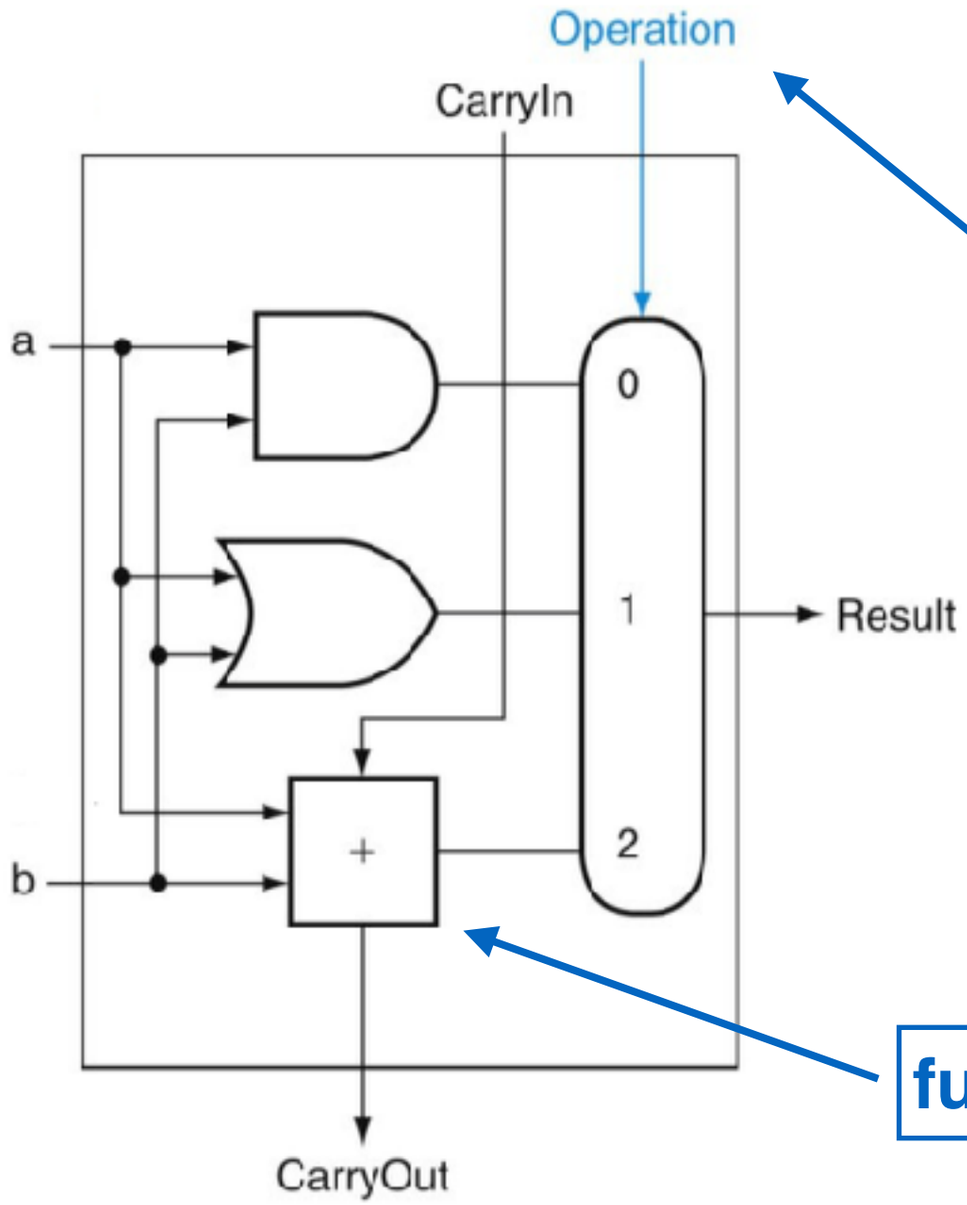
ALU - Full adder

- Il circuito che tiene conto del Carry In (riporto in input) si chiama “full adder”.
- E’ composto da due “half adder” in cascata più un OR che prende in input i due Carry Out.



1-bit ALU

- ALU a 1 bit che implementa le operazioni di somma, AND e OR:

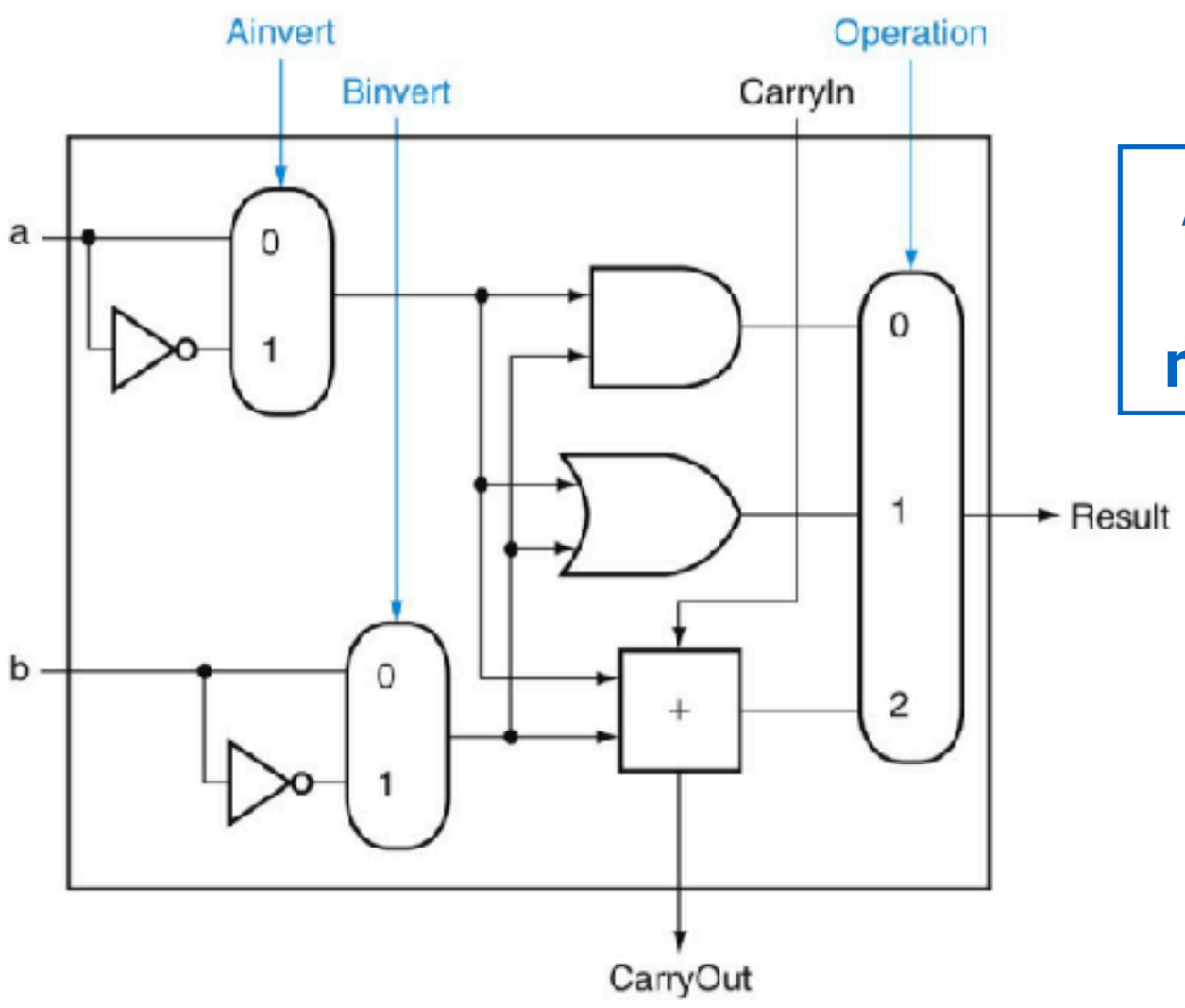


Operation è un multiplexer con 2 bit di controllo per scegliere l'operazione

full adder

1-bit ALU → inversione

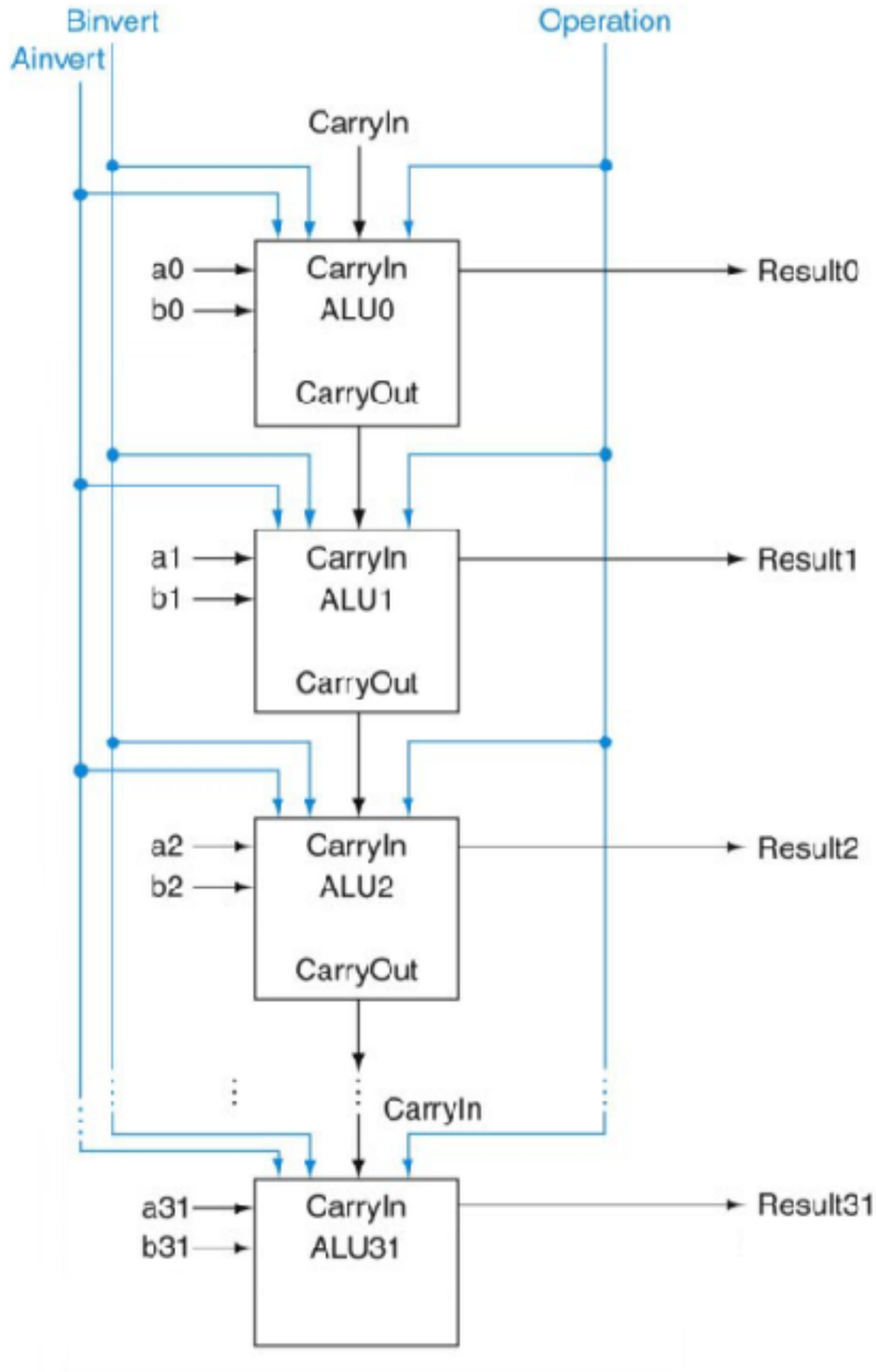
- Inversione degli input a e b
- Utile per:
  - $\overline{a + b} = \overline{a} \cdot \overline{b}$ (teorema di De Morgan)
  - $a - b = a + \overline{b} + 1$ (complemento a 2)



Ainvert e Binvert sono due multiplexer a 1 bit

32-bit ALU

- La ALU del processore MIPS deve gestire interi di 32 bit.
- Si mettono in array 32 ALU a 1 bit.



32-bit ALU → confronto

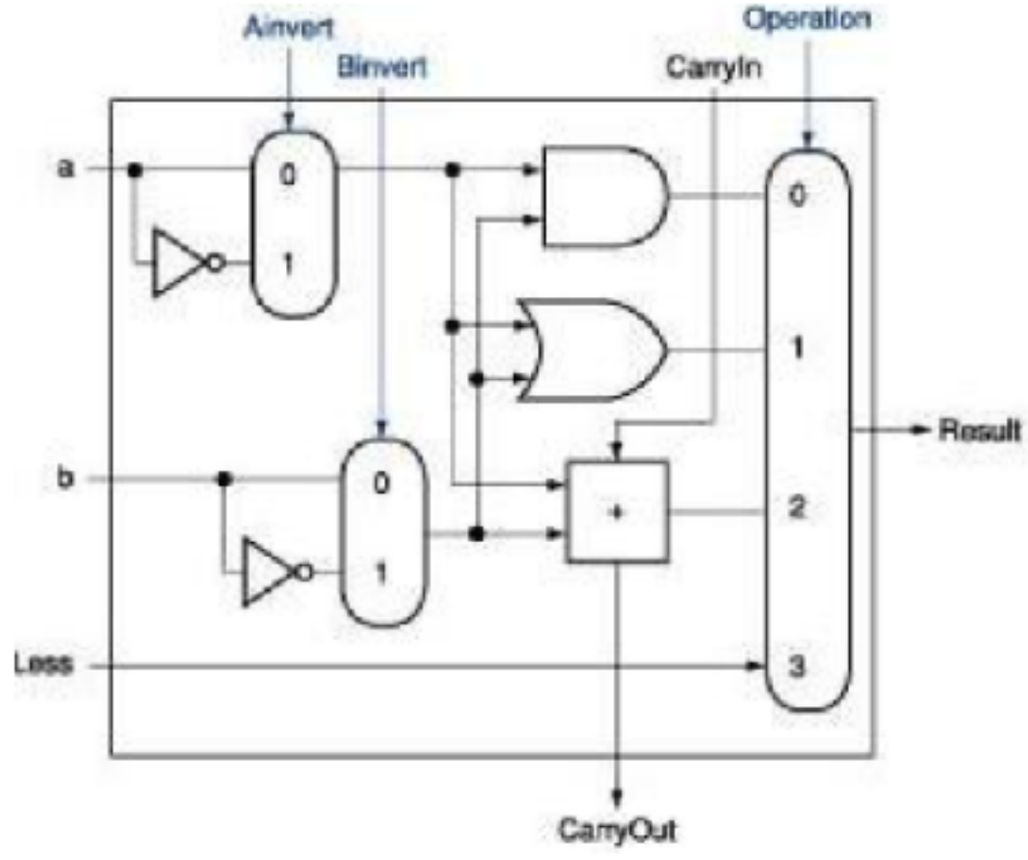
- Aggiungiamo il supporto hardware per eseguire il confronto:
  - istruzione slt (set-on-less-than)
    slt rd, rs, rt → se rs < rt allora rd = 1
  - istruzione beq (branch-on-equal)
    beq rs, rt, label → se rs = rt allora salta a label
- Soluzione: si effettua la sottrazione tra a e b (rs e rt).
- Se a - b è minore di zero allora a < b (per istruzione slt)
  - se a - b < 0 il risultato sarà 00….01, altrimenti sarà 00….00
- Se a - b è uguale a 0 allora a = b (per l'istruzione beq)
  - se a - b = 0 si deve mettere a 1 una linea di output utilizzata per l'uguaglianza (verrà aggiunta)

Istruzione slt

- E’ sufficiente vedere il bit più significativo (bit del segno) del risultato a - b.
- Problema:
  - se il bit più significativo del risultato di a -b vale 1, allora il bit meno significativo del risultato deve essere 1
  - se il bit più significativo del risultato di a - b vale 0, allora il bit meno significativo del risultato deve essere 0
- Quindi costruiamo l’ALU per tutti i bit eccetto il MSB (0 → 30) con un segnale Less aggiuntivo:
  - Attenzione: Less è un segnale di input, non di controllo del multiplexer!

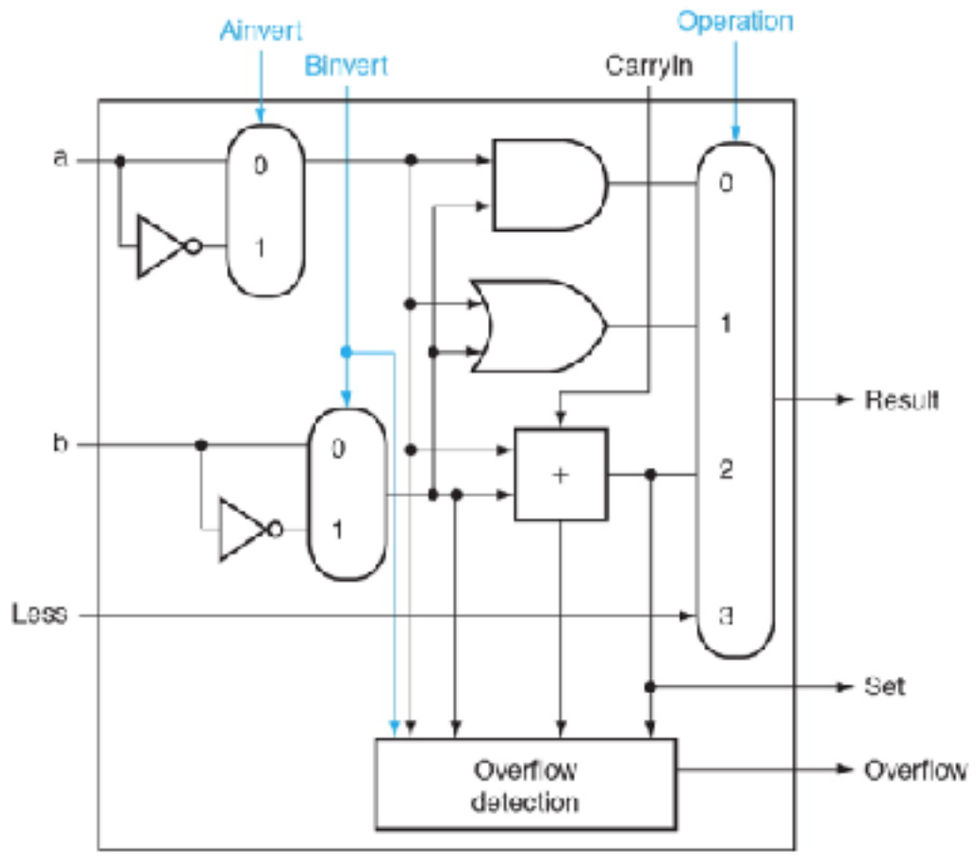
Istruzione slt: 0 → 30

- Soluzione per i bit da 0 a 30: se dal segnale di controllo Operation viene selezionata la linea 3 (Less) di input del multiplexer, il risultato diventa esattamente il valore di Less:
  - basta mettere 0 in ingresso a Less per ottenere il risultato voluto (eccezione fatta per il bit 0, ma lo vedremo nelle prossime slides)



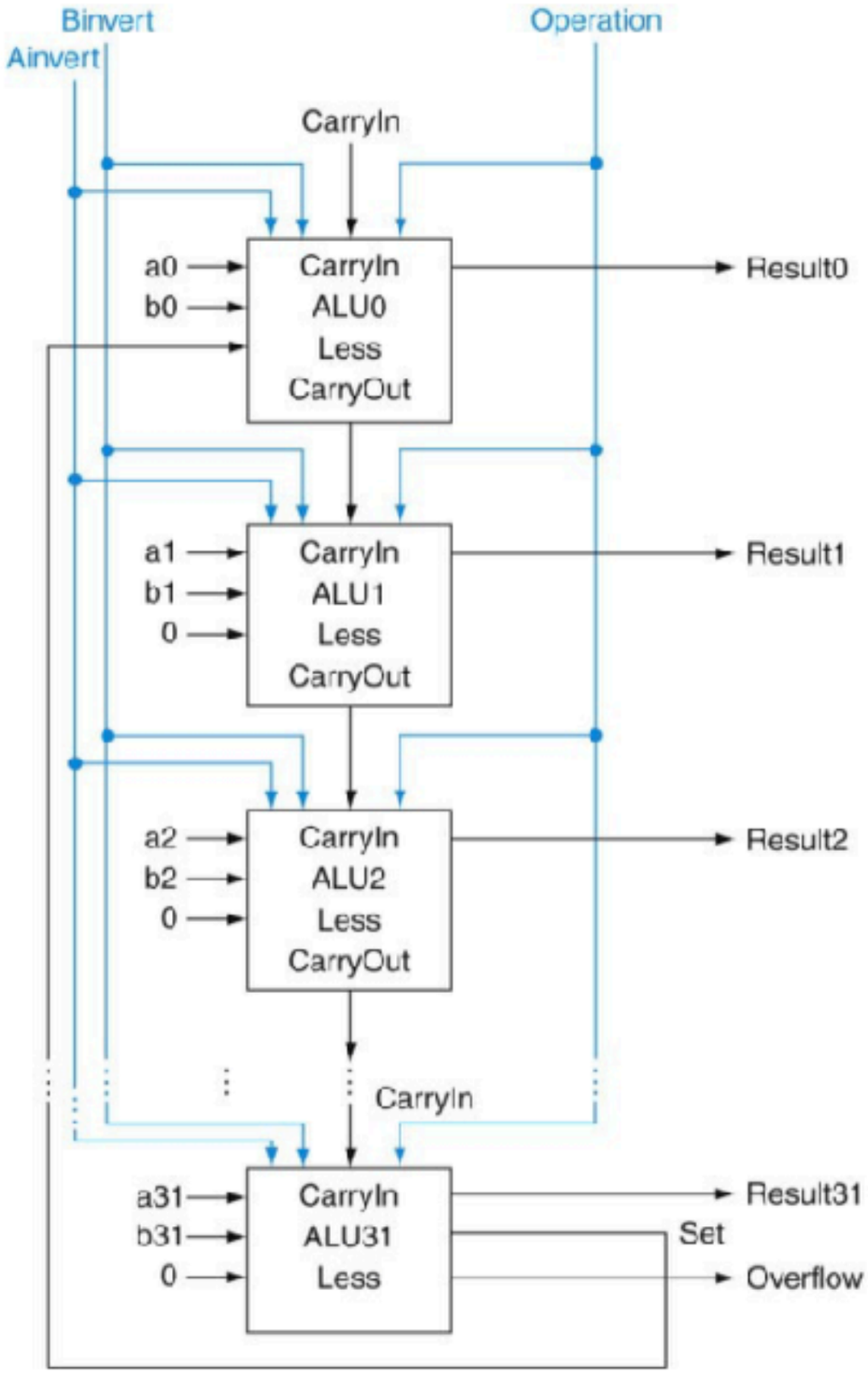
Istruzione slt: 31

- Il risultato del full adder (bit di segno) va sulla linea Set:
  - serve un output aggiuntivo, ossia il risultato del full adder (Set), che diventa l'input Less per la ALU del bit 0
- E' aggiunta anche una parte per determinare la condizione di Overflow.



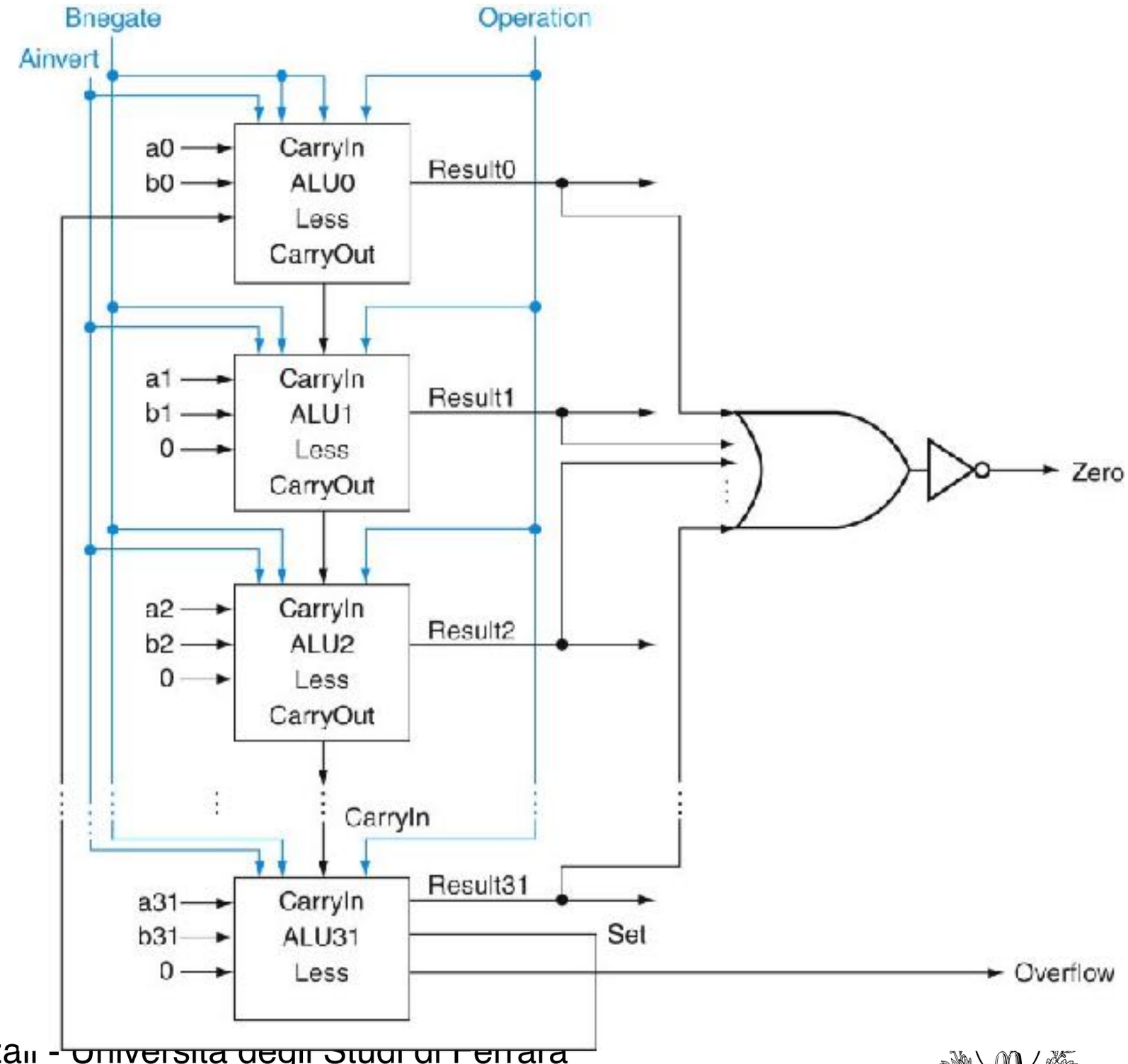
32-bit ALU

- Il bit più significativo di a - b viene usato per settare il bit meno significativo del risultato slt:
  - se a < b → a - b < 0 (numero negativo)



32-bit ALU

- Aggiungo il controllo se il risultato è uguale a 0.
- Notare che Bnegate controlla sia l'inversione dell'input b che il CarryIn della ALU per il bit meno significativo:
- $a - b = a + \overline{b} + 1$



32-bit ALU

- L’ALU è controllabile da 4 segnali:
  - Ainverse
  - Bnegate
  - 2 bit di selezione per il multiplexer Operation

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

32-bit ALU

- Simbolo usato per l'ALU (chiamata anche adder):