Architettura degli Elaboratori

Unità Aritmetico-Logica

Barbara Masucci
UNIVERSITA DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Punto della situazione

- Abbiamo studiato
  - Le reti logiche e la loro minimizzazione
  - I moduli combinatori di base utilizzati nei calcolatori

- Obiettivo di oggi
  - Studio dell'Unità Aritmetico Logica (ALU)

Arithmetic Logic Unit (ALU)

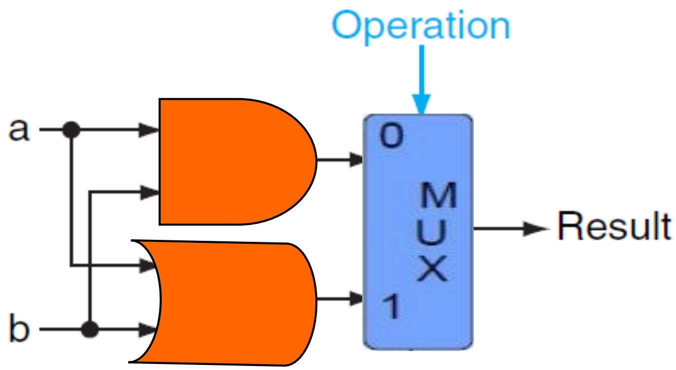
- Circuito combinatorio all'interno del processore per l'esecuzione di istruzioni macchina di tipo aritmetico/logiche:
  - Funzioni logiche: AND, OR, NOR
  - Funzioni aritmetiche:
    - Somma
    - Sottrazione
  - Istruzioni di confronto:
    - $a < b$?
    - $a = b$?
    - $a \neq b$?

ALU nel MIPS

- Nel processore MIPS, la parola macchina è a 32 bit
- Per gestire operazioni aritmetiche e logiche tra parole macchina a 32 bit, useremo 32 copie in cascata di una ALU a 1 bit
- Per costruire una ALU ad 1 bit, useremo un multiplexer 4:1 per scegliere l'operazione da eseguire
  - Sarà l'unità di controllo (CU) a settare opportunamente i due segnali di controllo del multiplexer 4:1

ALU a 1 bit

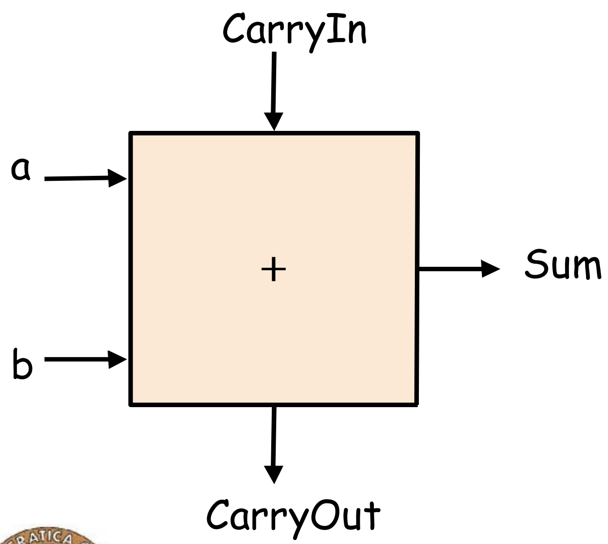
- Partiamo da una ALU ad 1 bit che supporti le operazioni di AND e OR tra due bit a, b
  - La scelta dell'operazione da eseguire può avvenire tramite un multiplexer 2:1
    - Se Operation = 0, Result = a AND b
    - Se Operation = 1, Result = a OR b



- Poi aggiungiamo un addizionatore a 1 bit per poter effettuare anche la somma

Addizionatore a 1 bit

- E' la componente che, su input tre bit a, b (i bit da sommare) e CarryIn (riporto in ingresso), dà in output
  - Il bit di somma (Sum)
  - Il riporto in uscita (CarryOut)



Inputs			Outputs	
a	b	CarryIn	CarryOut	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tavola di verità

Funzione CarryOut

Tavola di verità

a	b	CarryIn	CarryOut	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{a} \cdot b \cdot CarryIn$
1	0	0	0	
1	0	1	1	$a \cdot \bar{b} \cdot CarryIn$
1	1	0	1	$a \cdot b \cdot \overline{CarryIn}$
1	1	1	1	$a \cdot b \cdot CarryIn$

$$
\begin{aligned}
CarryOut &= \bar{a} \cdot b \cdot CarryIn + a \cdot \bar{b} \cdot CarryIn \\
&\quad + a \cdot b \cdot \overline{CarryIn} + a \cdot b \cdot CarryIn \\
&= (\bar{a} \cdot b + a \cdot \bar{b}) \cdot CarryIn + a \cdot b \cdot (\overline{CarryIn} + CarryIn) \\
&= (a \oplus b) \cdot CarryIn + a \cdot b
\end{aligned}
$$

Funzione Sum

Tavola di verità

a	b	CarryIn	Sum	
0	0	0	0	
0	0	1	1	$\overline{a}\cdot\overline{b}\cdot CarryIn$
0	1	0	1	$\overline{a}\cdot b\cdot\overline{CarryIn}$
0	1	1	0	
1	0	0	1	$a\cdot\overline{b}\cdot\overline{CarryIn}$
1	0	1	0	
1	1	0	0	
1	1	1	1	$a\cdot b\cdot CarryIn$

$$Sum = \overline{a}\cdot\overline{b}\cdot CarryIn + \overline{a}\cdot b\cdot\overline{CarryIn} + a\cdot\overline{b}\cdot\overline{CarryIn} + a\cdot b\cdot CarryIn$$

Ci vorrebbero 4 porte AND con tre input ciascuna!

Funzione Sum

$$\color{red}{Sum} = \bar{a}\cdot\bar{b}\cdot CarryIn + \bar{a}\cdot b\cdot\overline{CarryIn} + a\cdot\bar{b}\cdot\overline{CarryIn} + a\cdot b\cdot CarryIn$$

$$= (a\cdot\bar{b} + \bar{a}\cdot b)\cdot\overline{CarryIn} + (\bar{a}\cdot\bar{b} + a\cdot b)\cdot CarryIn$$

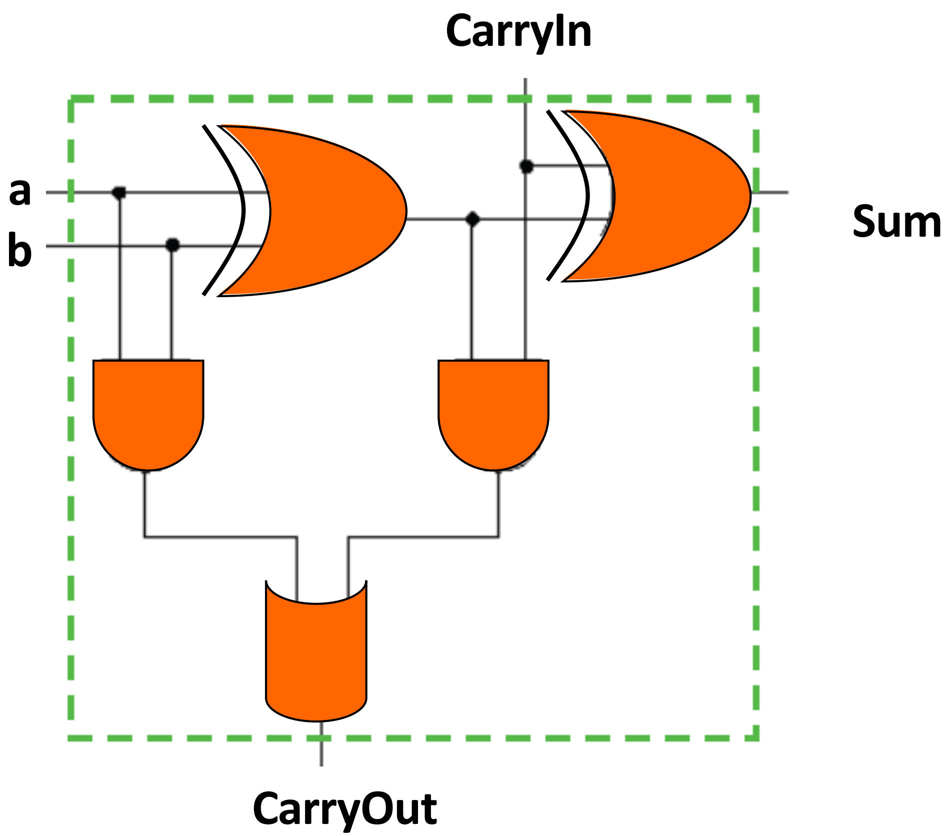
$$= (a \oplus b)\cdot\overline{CarryIn} + \overline{(a \oplus b)}\cdot CarryIn$$

$$= \color{red}{a \oplus b \oplus CarryIn}$$

Addizionatore a 1-bit con porte XOR

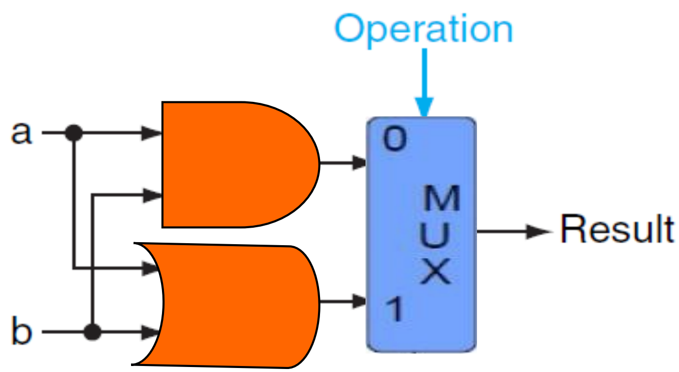
$$Sum = a \oplus b \oplus CarryIn$$

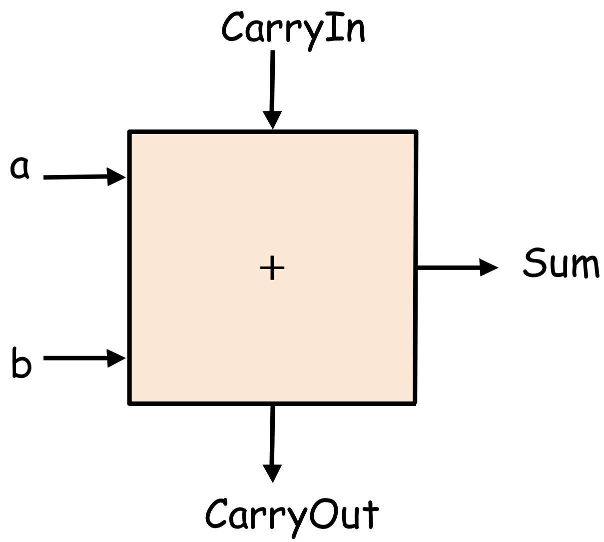
$$CarryOut = (a \oplus b) \cdot CarryIn + a \cdot b$$

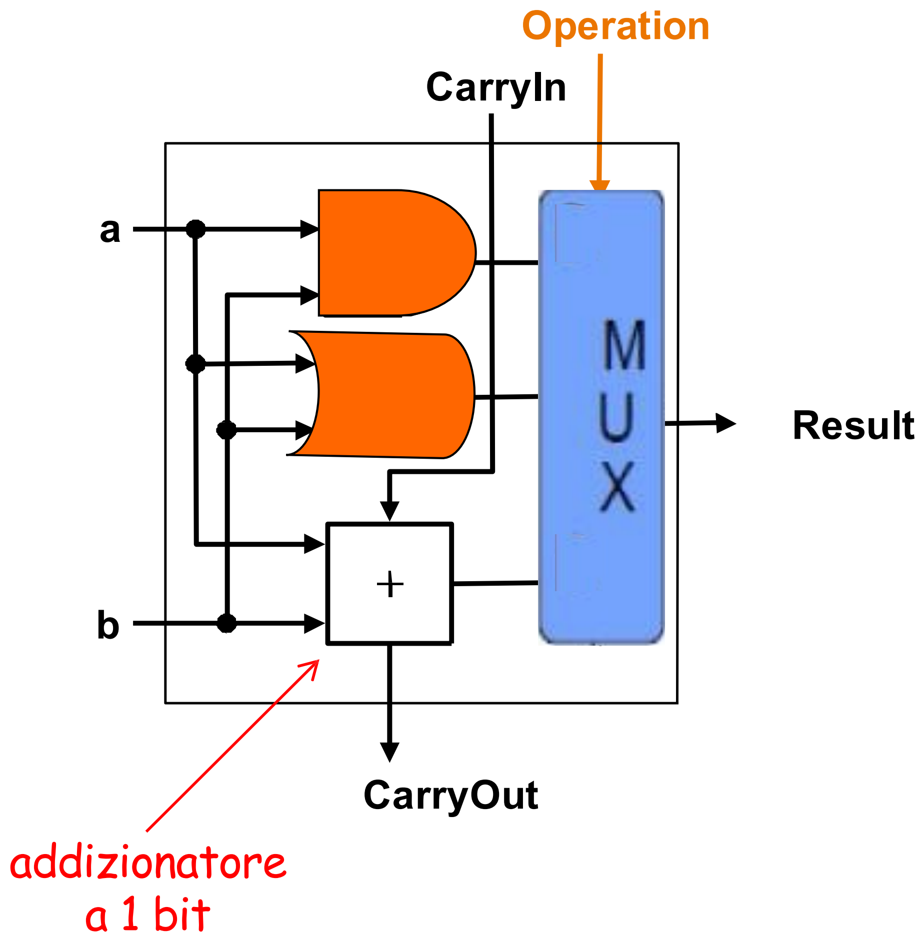


ALU a 1 bit

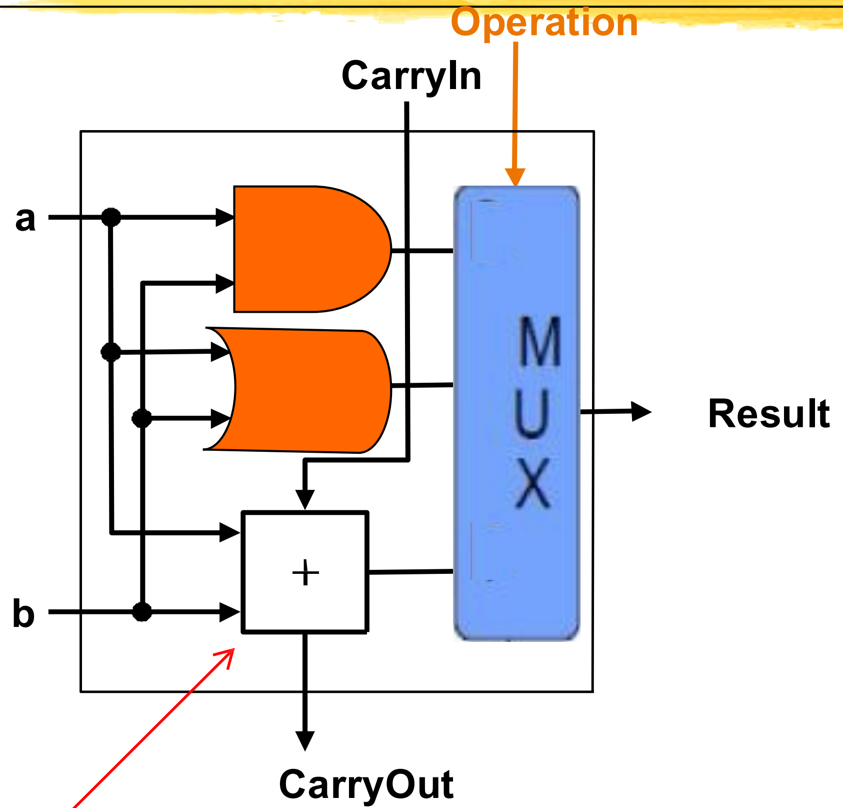
- Aggiungiamo l'addizionatore a 1 bit all'ALU che supporta AND e OR





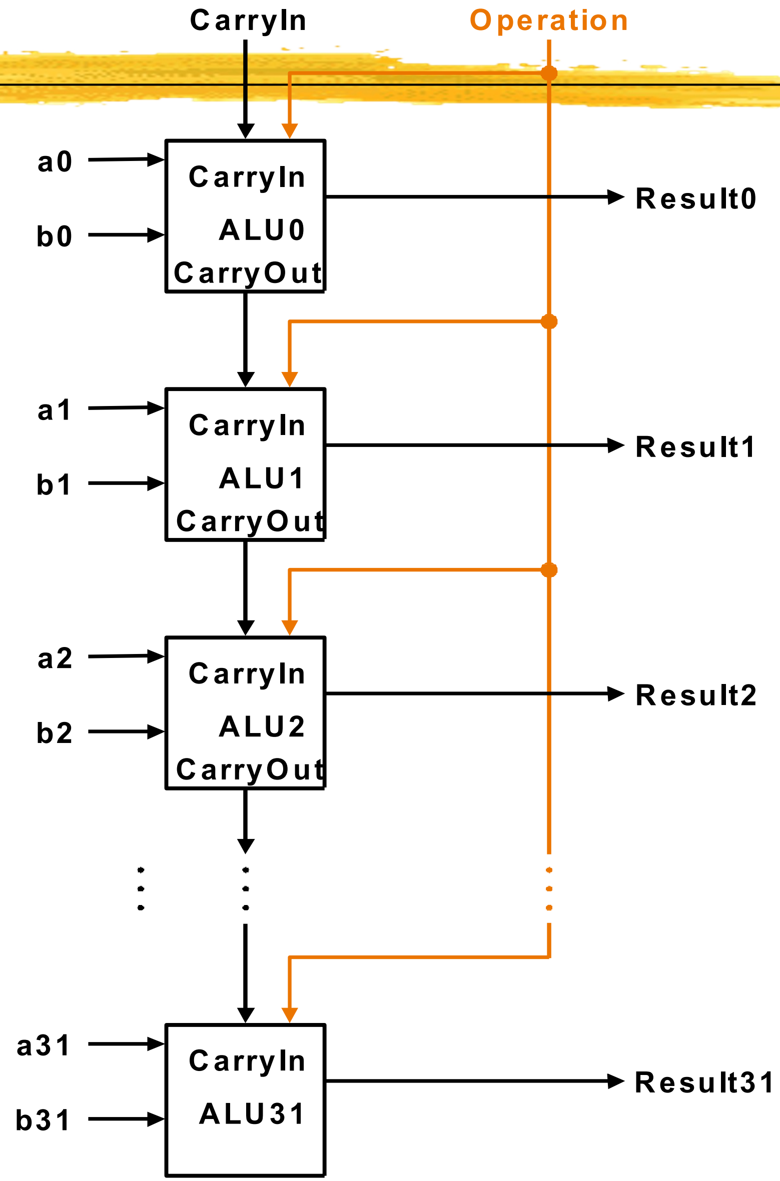


ALU a 32 bit



addizionatore
a 1 bit

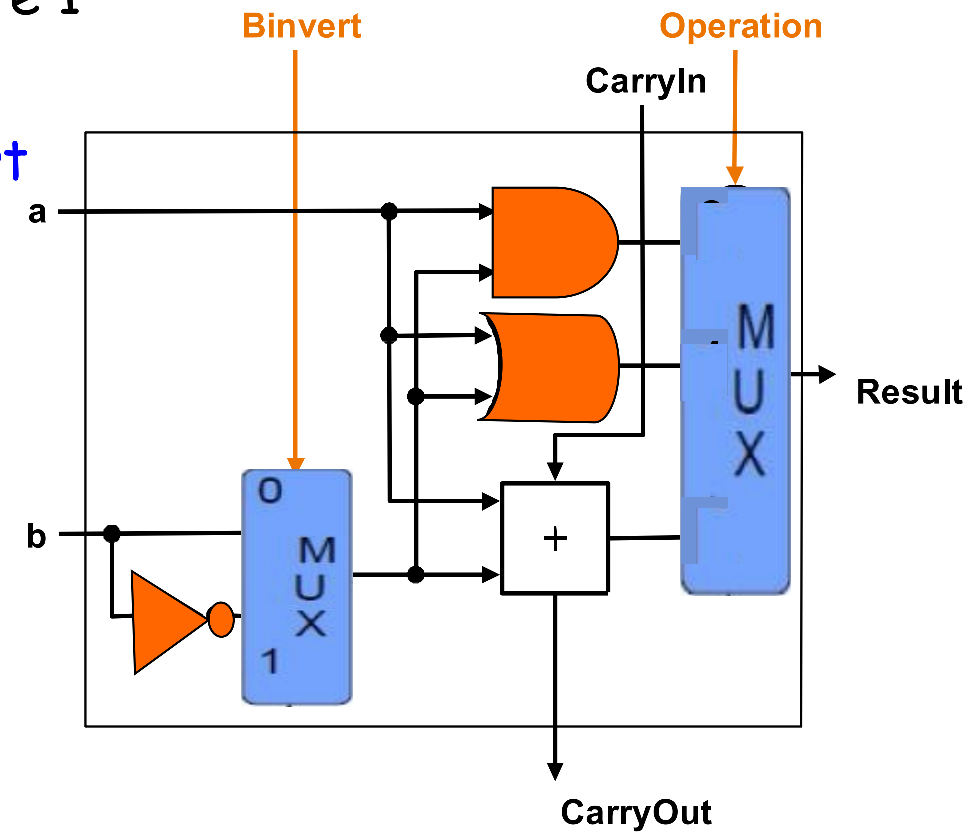
Operation	Istruzione
00	and
01	or
10	sum



Calcolare a – b

- La differenza a - b si ottiene calcolando a + (-b)
  - Per ottenere l'opposto di b (in complemento a 2), possiamo invertire i bit di b e sommare 1
- Una buona soluzione:
  - Segnale di controllo Binvert
  - Porre a 1 CarryIn di ALU0

Binvert	Operation	Istruzione
0	00	and
0	01	or
0	10	sum
1	10	sub



Il confronto: a < b?

- Dobbiamo supportare anche una istruzione di confronto (slt: set if less than) tra due numeri a, b di 32 bit
  - Il risultato del confronto è 1 se a < b, 0 altrimenti
    - set Result to 1 if a is less than b
  - Più precisamente, l'output del confronto è una stringa a 32 bit:
    - Result = 00..01 se a < b,
    - Result = 00..00 altrimenti
- Per stabilire se a < b sfrutteremo la possibilità di effettuare la sottrazione a-b:
  - Se otterremo un numero negativo come risultato della sottrazione a - b, allora a < b

Il confronto: a < b?

- Vogliamo che
  - se a < b allora Result = 00..01 (solo un 1 su 32 bit)
  - se a >= b allora Result = 00..00 (tutti 0 su 32 bit)

  Quindi, sempre: $Result_{31}=...= Result_2= Result_1=0$

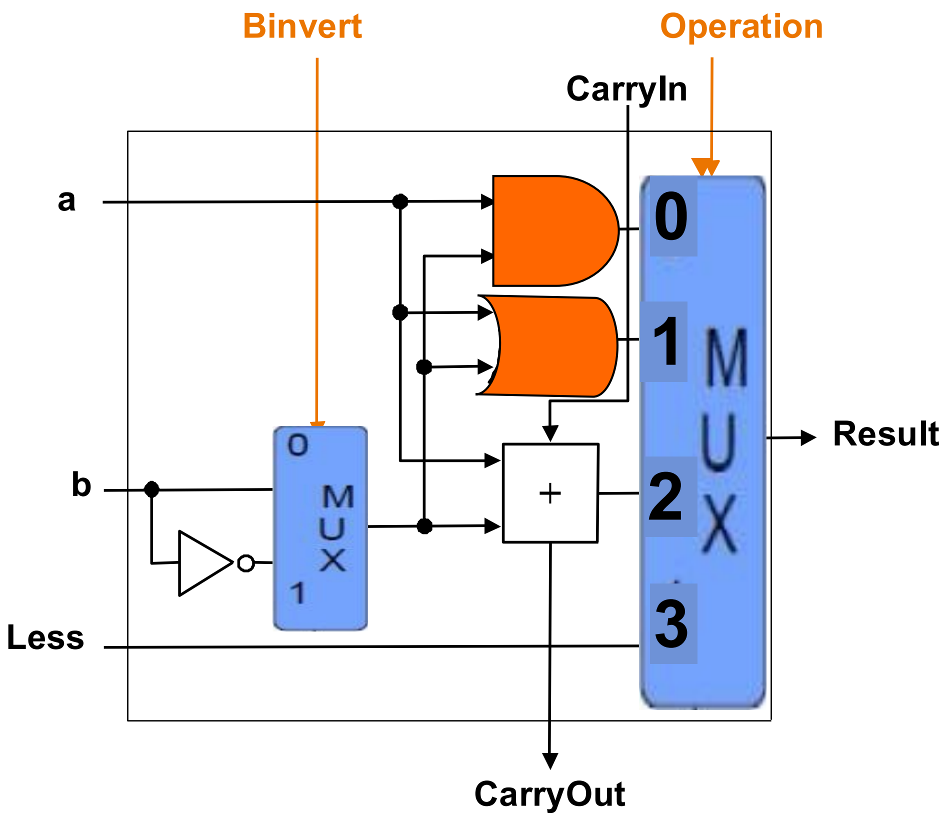
- Inoltre, vogliamo che
  - se a < b allora $Result_0 = 1$
  - se a >= b allora $Result_0=0$

  Eseguiamo la sottrazione a-b in complemento a 2 e chiamiamo Set il bit più significativo (bit 31) della differenza ($Set=Sum_{31}$ viene dato in output da ALU31)
  - Se a < b allora a – b < 0, quindi Set sarà 1
  - Se a >= b allora a – b >= 0, quindi Set sarà 0

Quindi basta porre $Result_0=Set$

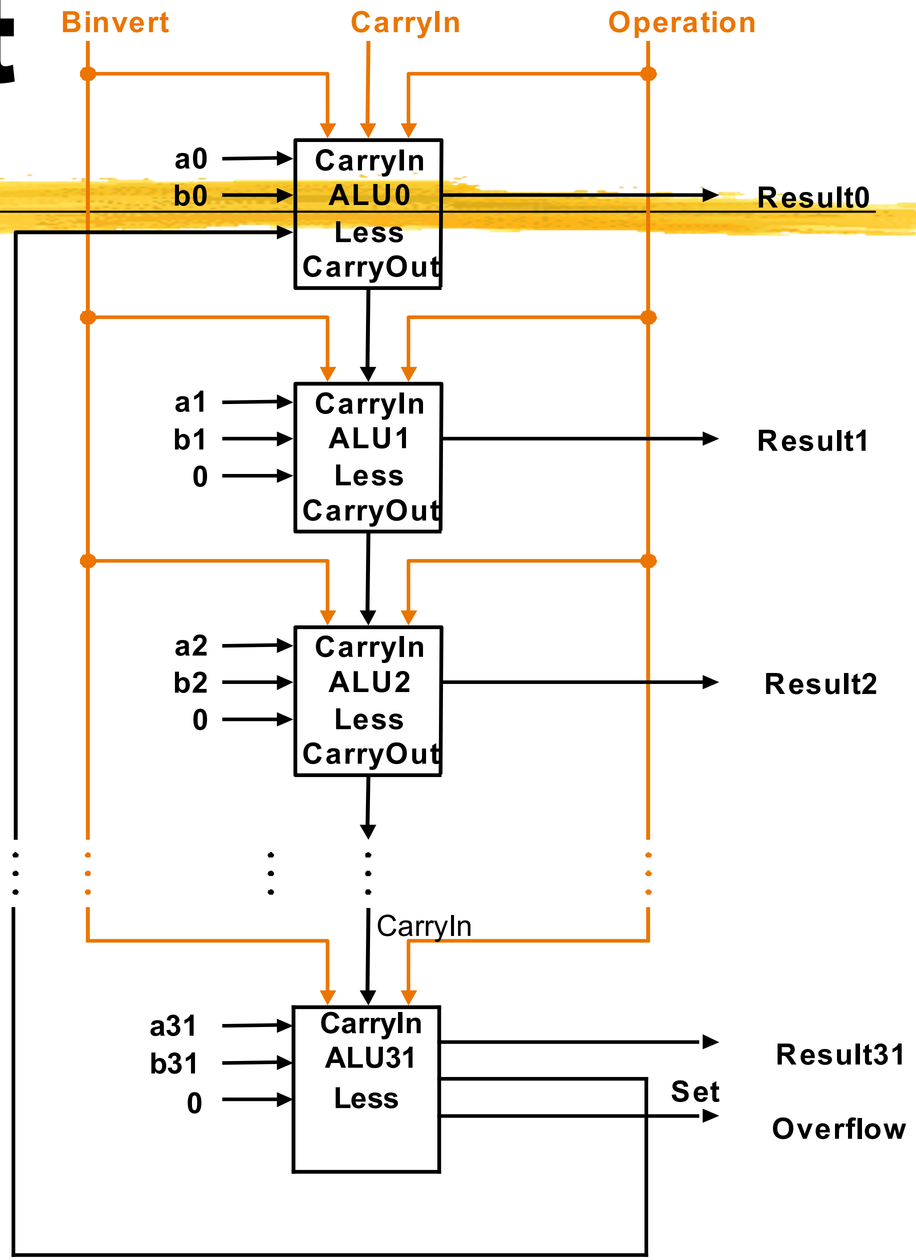
Il confronto: a < b?

- Definiamo un nuovo ingresso Less a 1 bit per ogni ALU



Supportare slt

- L'output Set di ALU31 viene ridiretto sull'input Less di ALU0
- Gli input Less delle varie ALU (eccetto la prima) sono posti a 0
- Binvert e CarryIn sono posti a 1 per sottrarre (nelle istruzioni sub e slt)
- Operation è posta a 11 per far passare in output l'ultimo bit in ingresso al multiplexer 4:1



Binvert	Carryin	Operation	Istruzione
0	0	00	and
0	0	01	or
0	0	10	sum
1	1	10	sub
1	1	11	slt

Sono sempre uguali: possono essere unificati in un unico segnale di controllo

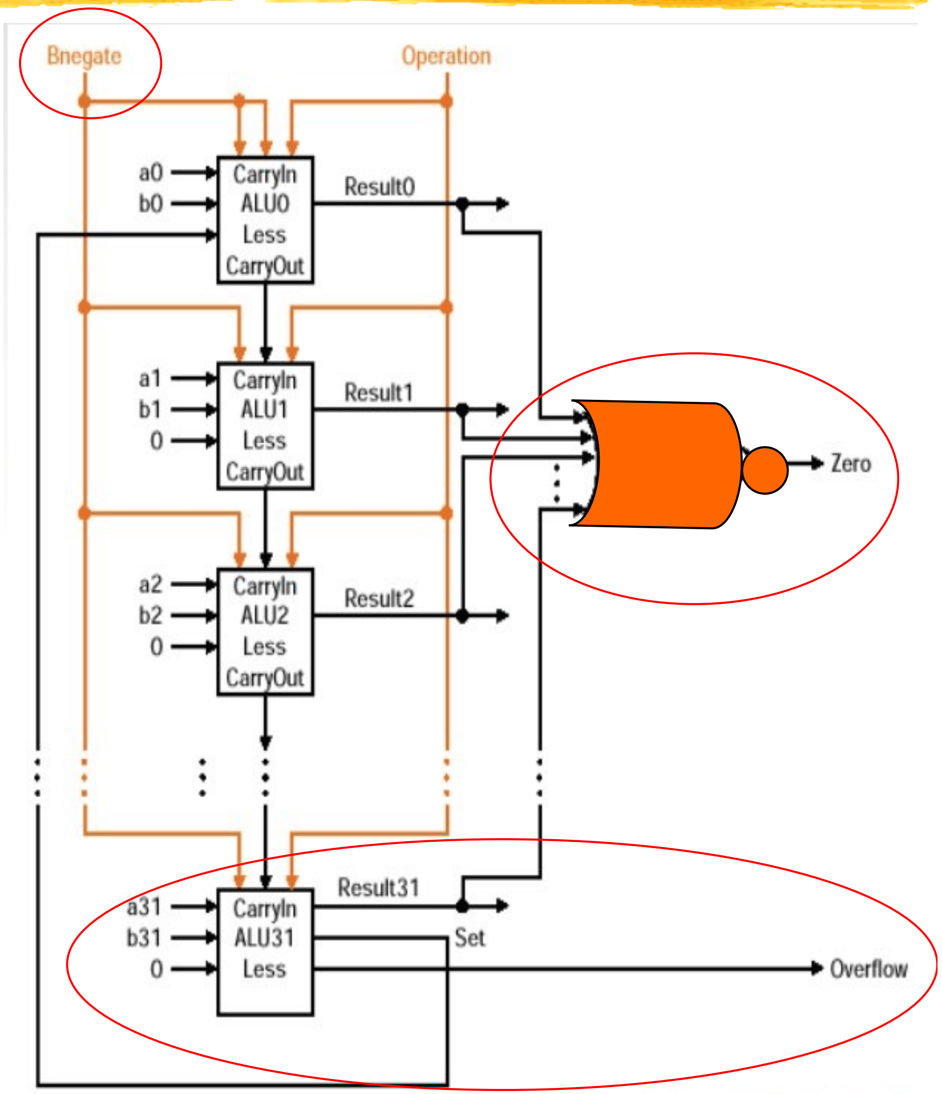
Test per l'uguaglianza

- Il modo più facile per supportare il test di uguaglianza (a = b?) è testare se $a - b = 0$
- Quindi:
  - Sottrarre b da a
  - Testare se il risultato è 0
- Per testare se il risultato (Result) è 0, generiamo un nuovo output per l'ALU: il bit Zero
  - Zero = 1 se Result =0
  - Zero = 0 altrimenti
  - Nota che: $Zero = \overline{Result_{31} + Result_{30} + \ldots + Result_1 + Result_0}$

OR

ALU quasi finale

- L'aggiunta dell'output Zero (generato da una porta NOR) consente di supportare le istruzioni beq e bne
  - Zero=1 se e solo se a=b
- Dato che Binvert e CarryIn sono sempre uguali possono essere unificati in un unico segnale di controllo: Bnegate
- All'ALU31 è stato aggiunto anche l'output Overflow (1 bit)



Supportare NOR

- E' necessario prevedere un insieme funzionalmente completo di operatori logici
  - Per poter realizzare qualsiasi funzione booleana

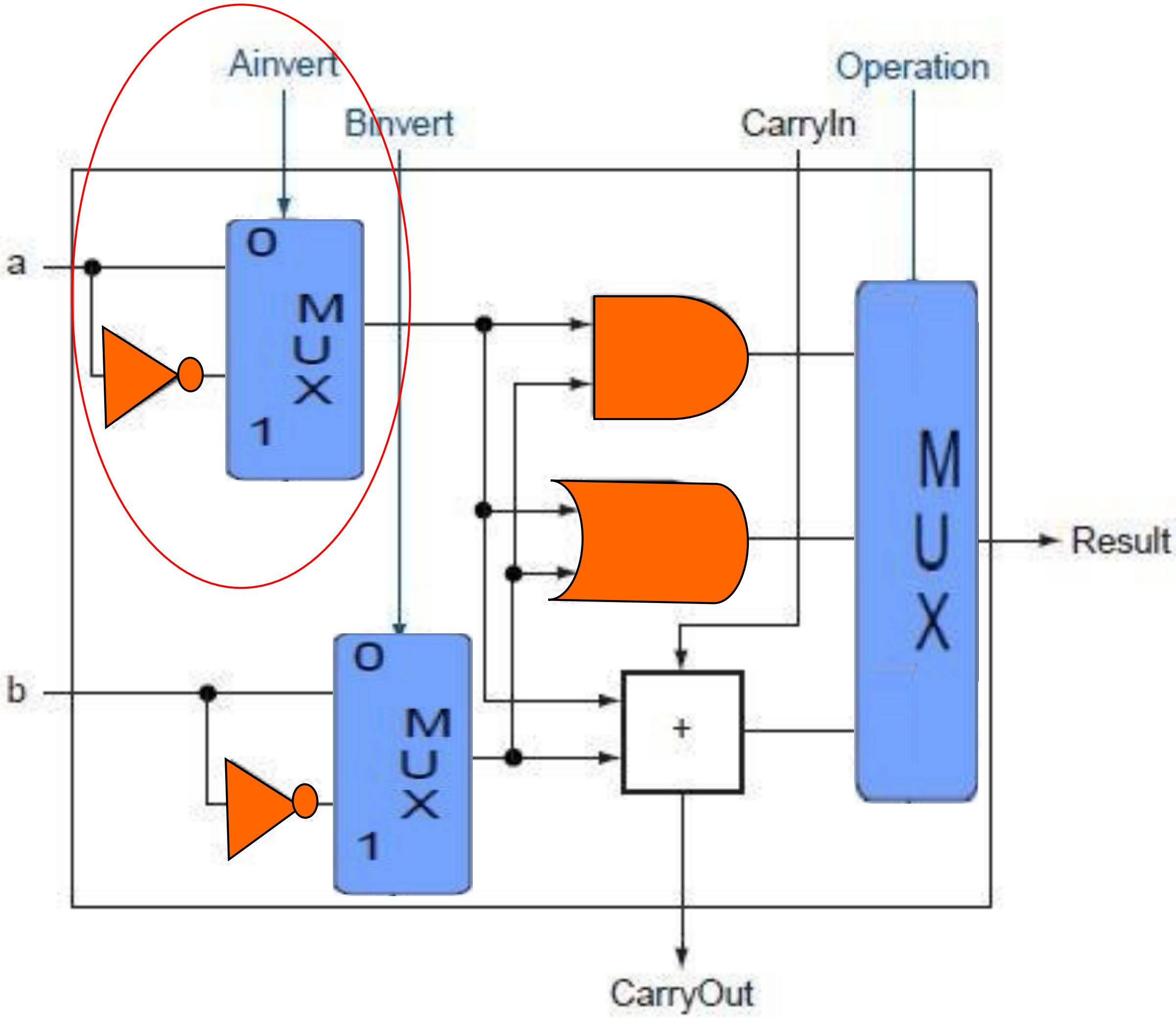
- La modifica di minore impatto al progetto dell'ALU consiste nell'introdurre la funzione NOR(a,b) attraverso la legge di De Morgan:

  NOR(a,b) = NOT(OR(a,b)) = AND(NOT(a),NOT(b))

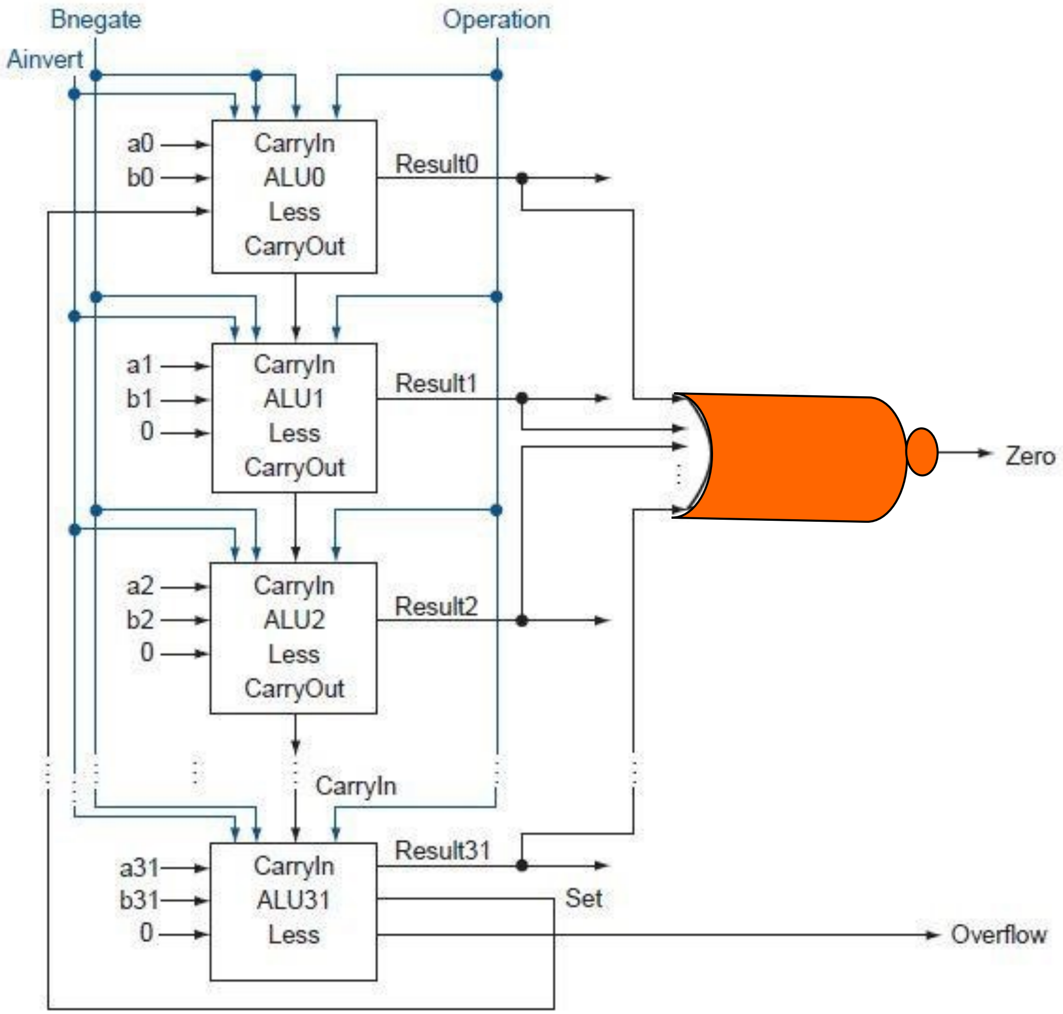
- Modifica necessaria:
  - Dobbiamo invertire anche a

- Analogamente, possiamo realizzare anche NAND

Supportare NOR

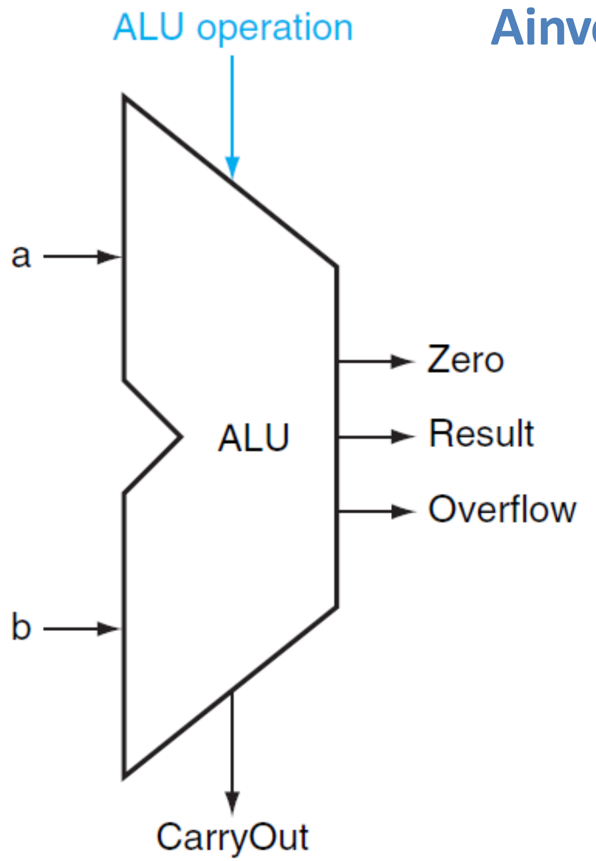


ALU finale



Bnegate
Operation
Ainvert
a0
b0
CarryIn
ALU0
Less
CarryOut
Result0
a1
b1
0
CarryIn
ALU1
Less
CarryOut
Result1
Zero
a2
b2
0
CarryIn
ALU2
Less
CarryOut
Result2
CarryIn
a31
b31
0
CarryIn
ALU31
Less
Result31
Set
Overflow

Simbolo per ALU nella CPU



Ainvert (1 bit), Bnegate (1 bit), Operation (2bit)

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

Riepilogo e riferimenti

- Abbiamo costruito la componente che nella CPU si occuperà di: AND, OR, NOR, somma, sottrazione, confronto e test per l'uguaglianza
  - [PH] Appendice B.5