


La ALU

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano

Riferimento sul Patterson: sezione C.5


Sommario

ALU ad 1 bit

ALU a 32 bit

Comparazione, Overflow, Test di uguaglianza

Tecnologie di costruzione di una ALU

Funzione della ALU

E' integrata nel processore, all'inizio degli anni 90 è stata rivoluzionaria la sua introduzione con il nome di co-processore matematico.

Esegue le operazioni aritmetico-logiche.

E' costituita da circuiti combinatori. Utilizza i blocchi di base già visti.

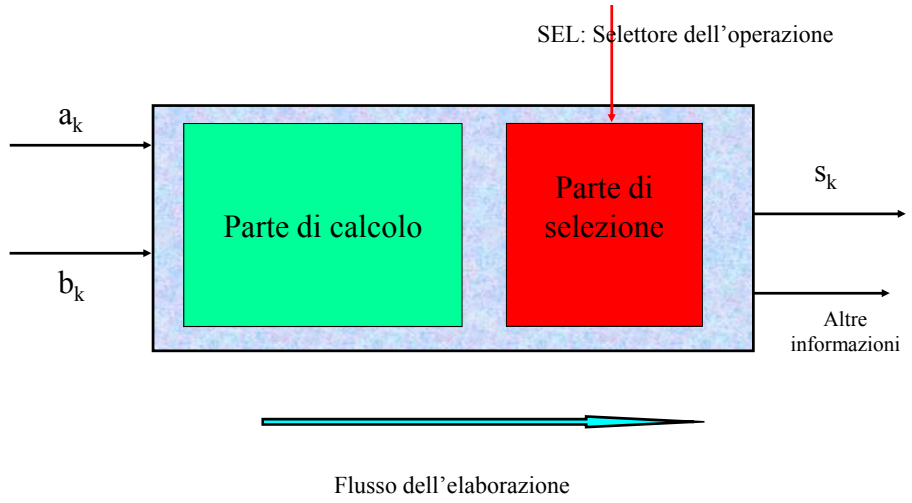
Opera su parole (MIPS 32 bit).

Le ALU non compaiono solamente nei micro-processori.

Problematiche di progetto

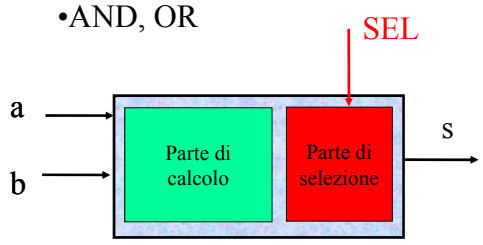
- Velocità (Riporto).
- Costo.
- Precisione.
- Affidabilità
- Consumo.

Struttura a 2 livelli di una ALU

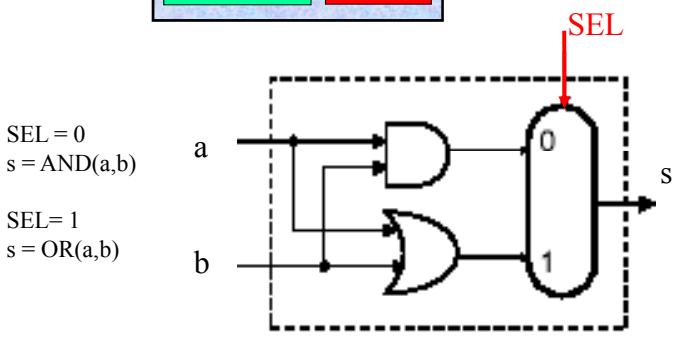


Progettazione della ALU – 1 bit

•AND, OR



a b	a *AND* b	a *OR* b
0 0	0	0
0 1	0	1
1 0	0	1
1 1	1	1



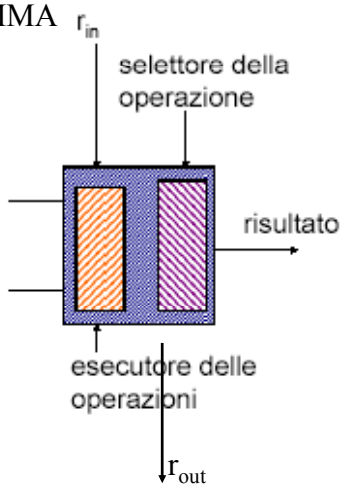
SEL = 0
s = AND(a,b)

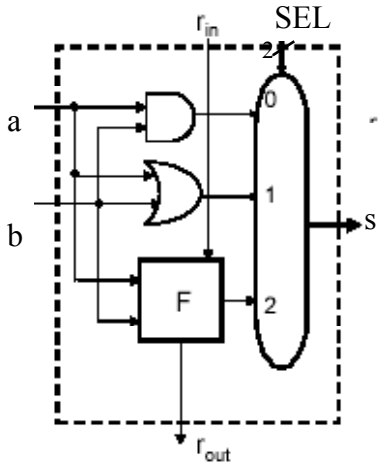
SEL= 1
s = OR(a,b)

1 porta AND
1 porta OR
1 Mux

La nuova struttura della ALU – 1 bit

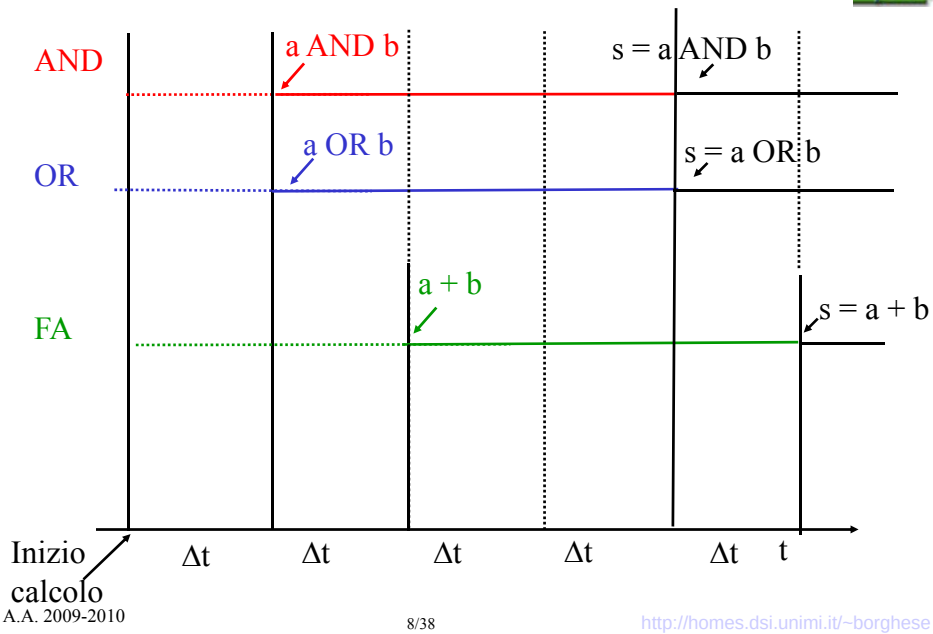
- AND
- OR
- SOMMA





Perchè SEL non viene messo in ingresso?

I cammini critici all'interno della ALU



Sommario

ALU ad 1 bit

ALU a 32 bit

Comparazione, Overflow, Test di uguaglianza

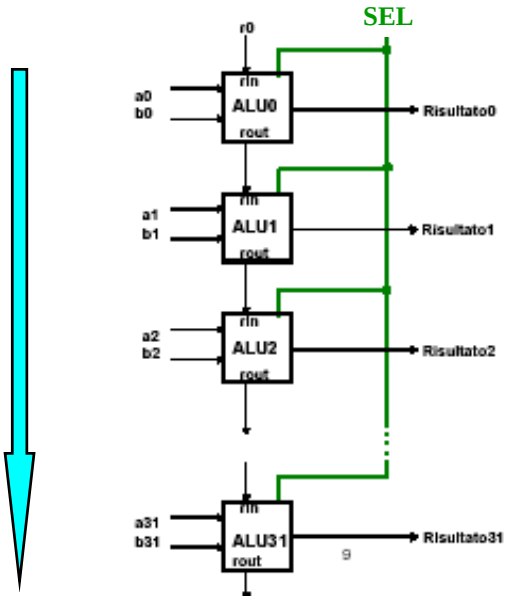
Tecnologie di costruzione di una ALU

ALU a 32 bit

Come collegare le ALU ad 1 bit?

Flusso di calcolo

Perchè non si può parallelizzare?



Sottrazione

In complemento a 2 diventa un'addizione: $a - b = a + \overline{b} + 1$

Esempio: s = 3 – 4; su 3 bit

3 -> 011	011 +
-4 -> 100 in complemento a 2	100 =
-1 -> 111 in complemento a 2	111

Posso scrivere il numero negativo in complemento a 2 come somma:

	4 -> 100	numero positivo:	b
Passo I – Complemento a 1	011+	complemento a 1:	$\overline{b}$+
Passo II – Sommo + 1	1=	sommo 1:	1=
Risultato – Complemento a 2	100	risultato	-b

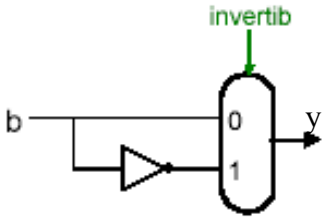
Posso quindi scrivere: $-b = \overline{b} + 1$

Sottrazione

In complemento a 2 diventa un'addizione: $a - b = a + \overline{b} + 1$

Serve:
a) un inverter (NOT).
b) la costante 1

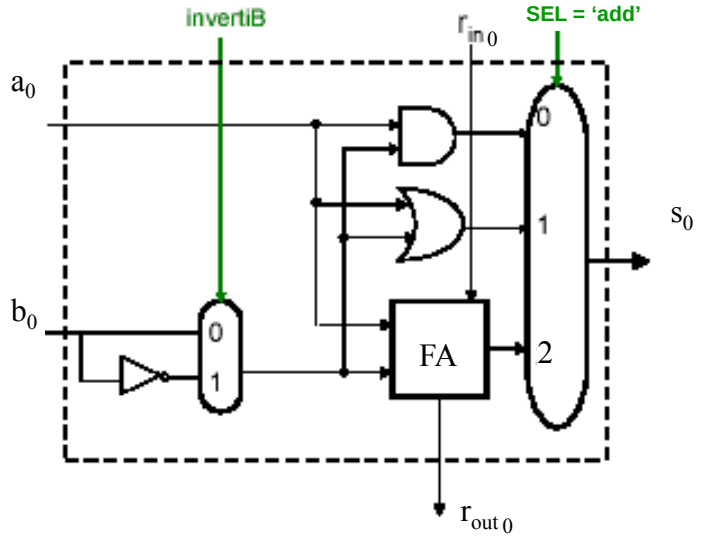
a)



Iff invertib
y = !b

b) Da dove prendo la costante 1?

Sottrazione - ALU_0

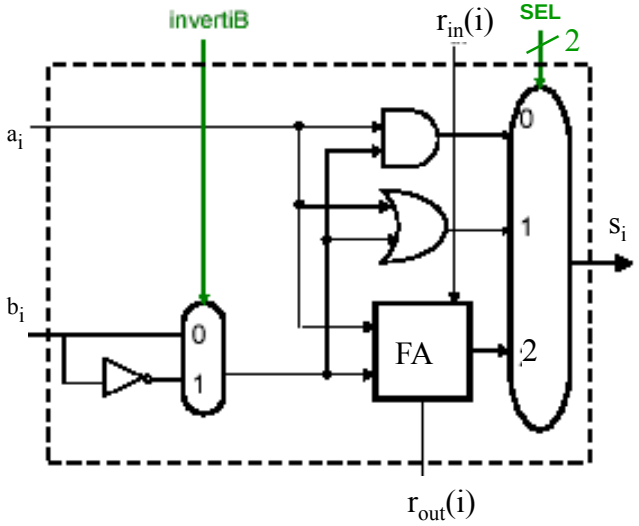


$r_{in}(0)$ = InvertiB = 1 se sottrazione

(occorre utilizzare un full adder anche per il bit meno significativo con $r_{in0} = 1$).

Sottrazione - ALU_i

- AND
- OR
- SOMMA
- SOTTRAZIONE



$r_{in}(i) = r_{out}(i-1)$ $i = 1,2,3,....31$ $i \neq 0$

InvertiB = 1 se sottrazione

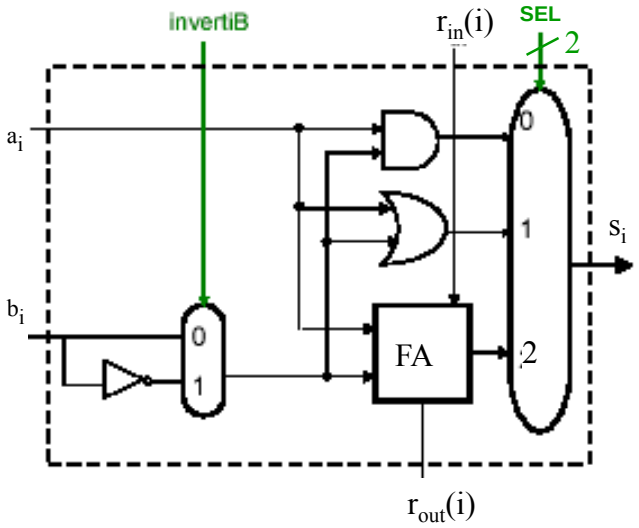
Operazioni particolari - ALU_i

E' possibile programmare questa ALU per eseguire

a AND !b

oppure:

a OR !b



InvertiB = 1
SEL = AND, OR

Sottrazione: ALU a 32 bit

$r_{in}(0)$ = InvertiB = 1
se sottrazione

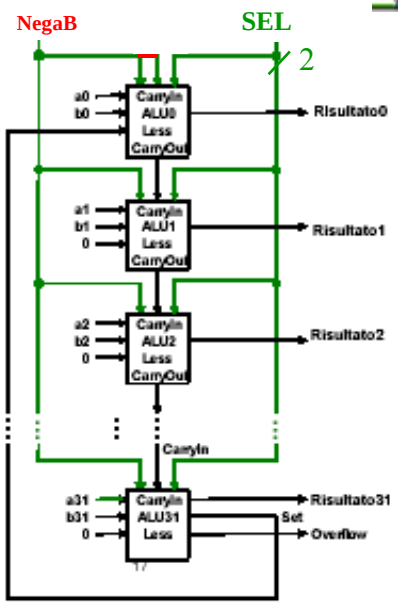
- AND
- OR
- SOMMA
- SOTTRAZIONE

From_UC	SEL	r_0	InvertiB
And	And	0	0
Or	Or	0	0
Somma	Add	0	0
Sottr.	Add	1	1

InvertiB e r_0 sono lo stesso segnale, si può ancora ottimizzare.

$r_{in}(0)$ entra solo in ALU_0

InvertiB entra in tutte le ALU_i



Sommario

ALU ad 1 bit

ALU a 32 bit

Comparazione, Overflow, Test di uguaglianza

Tecnologie di costruzione di una ALU

Confronto

Fondamentale per dirigere il flusso di esecuzione (test, cicli....)

if a less_than b then
s = 1

if a < b then
s = 1

if (a – b) < 0 then
s = 1

Come sviluppare la comparazione - I?

if **(a – b) < 0** then
s = 1
else
s = 0

if **differenza(a,b) < 0** then
s = 1
else
s = 0

Si controlla che il primo bit del risultato della somma (bit di segno) sia = 1.

Esempio: s = 3 – 4; su 3 bit

3 -> 011
-4 -> 100 in complemento a 2
-1 -> 111 in complemento a 2

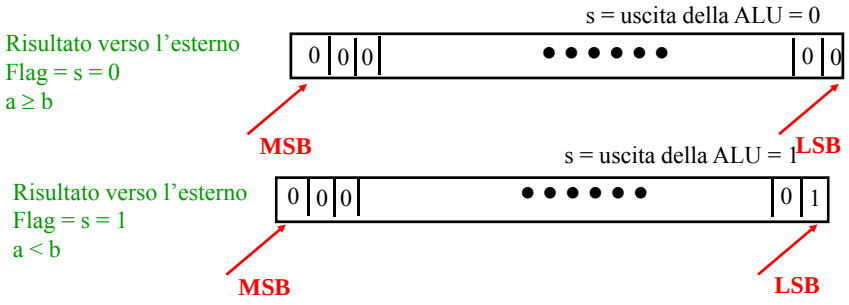
0 1 1 +
1 0 0 =
1 1 1

MSB = bit di segno

Come sviluppare la comparazione - II?

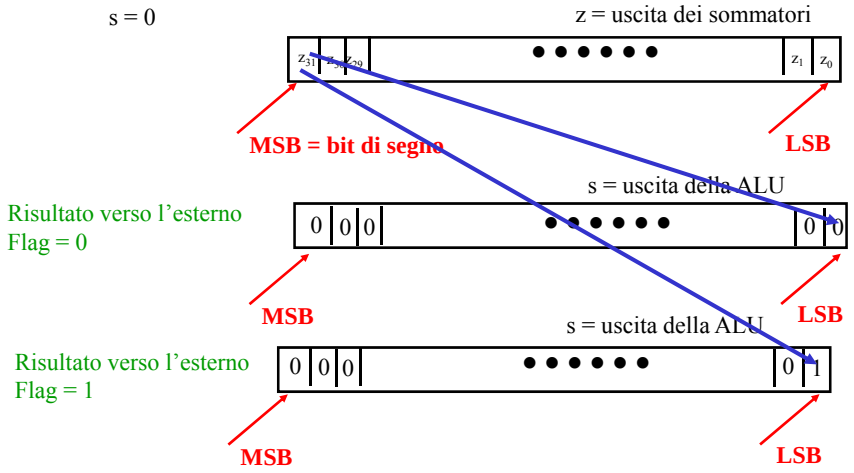
if (a – b) < 0 then
s = 1
else
s = 0

Viene impostato un flag = 1 se a < b.
Valori possibili in uscita della ALU: {0, 1}



Come sviluppare la comparazione (riassunto)?

if (a – b) < 0 then
s = 1
else
s = 0

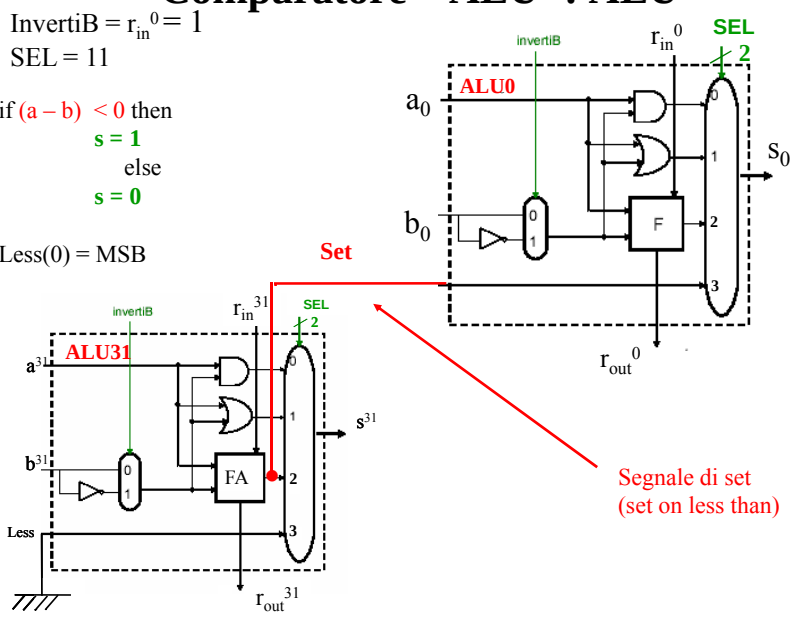


Comparatore – ALU^0 : ALU^{31}

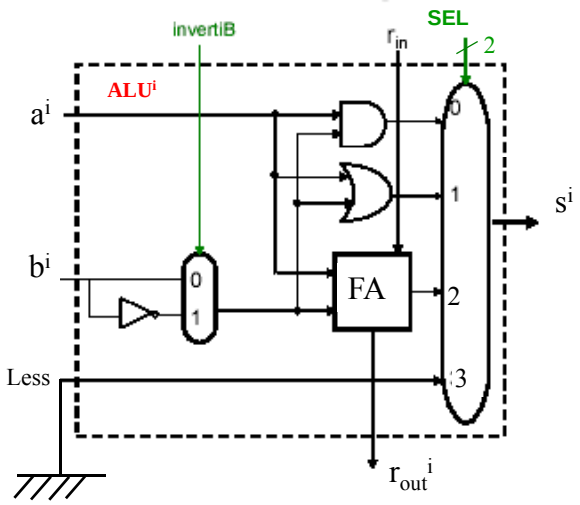
InvertiB = $r_{in}^0 = 1$
SEL = 11

if (a – b) < 0 then
s = 1
else
s = 0

Less(0) = MSB



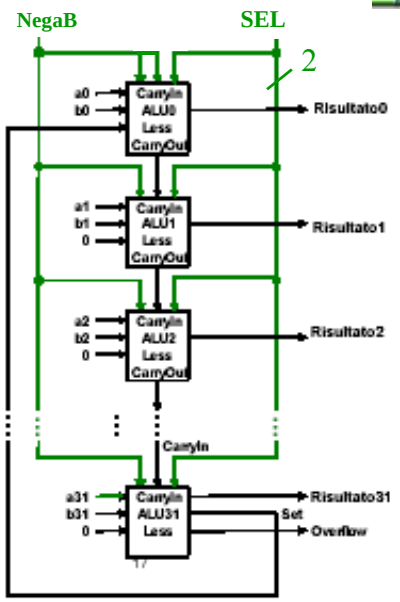
Comparatore - ALU^i



Less(i) = 0 i = 1,2,3, ...31 i ≠ 0

Comparazione - ALU completa a 32 bit

- AND
- OR
- SOMMA
- SOTTRAZIONE
- COMPARAZIONE



Overflow decimale

a + b = c - codifica su 2 cifre,

a = 12, b = 83 => Not Overflow.

012+
083=

095

a= 19, b = 83 => Overflow

019+
083=

102

Si può avere overflow con la differenza?

Overflow binario

a + b = c - codifica su 4 cifre binarie, di cui 1 per il segno.

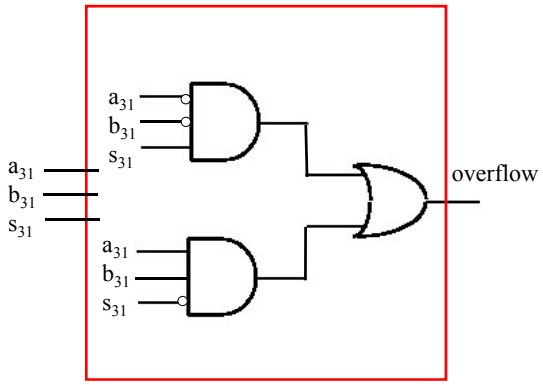
No overflow		Overflow	
0010+	2+	0010+	2+
0011=	3=	0111=	7=
------	----	-------	---
0101	5	1001	-7??
1010+	-6+	1010+	-6+
1111=	-1=	1011=	-5=
------	----	-------	---
1001	-7	0101	+5??

Quindi si ha overlow nella somma quando:
a + b = s, a > 0, b > 0 MSB di a e b = 0, MSB di s = 1.
a + b = s, a < 0, b < 0 MSB di a e b = 1, MSB di s = 0.

Circuito di riconoscimento dell'overflow

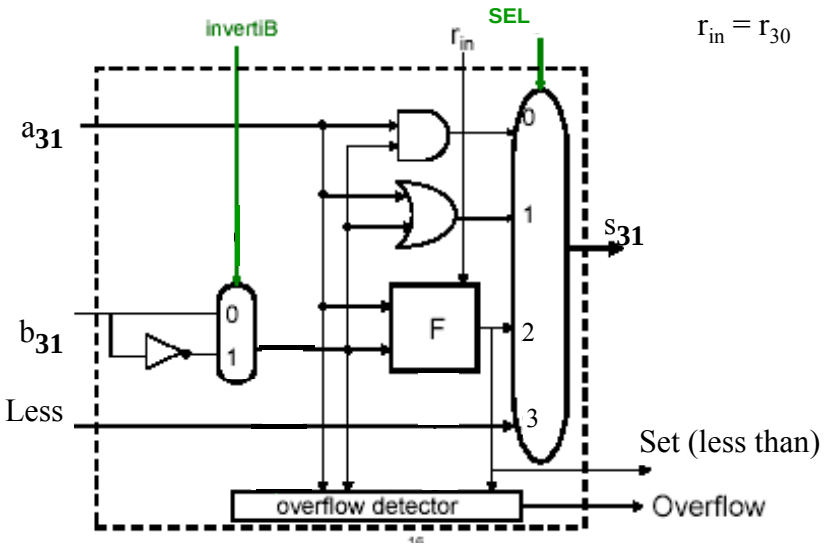
Lavora sui MSB

a_{31}	b_{31}	s_{31}	overflow
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



Overflow detector

ALU_{31}



$r_{in} = r_{30}$

Operazione di uguaglianza

beq rs, rt label iff (rs – rt) = 0 , salta.

Occorre quindi:
- Impostare una differenza.
- Effettuare l'OR logico di tutti i bit somma.
- L'uscita dell'OR logico = 0 i due numeri sono uguali.

Input: a, b su 32 bit
Output: {0,1}. In questo caso viene dedicata una linea di uscita.

Uguaglianza – prima possibilità

beq rs, rt label iff (rs – rt) = 0 , salta.

A_0	B_0	C_0
0	0	1
0	1	0
1	0	0
1	1	1

A_1	B_1	C_1
0	0	1
0	1	0
1	0	0
1	1	1

…

$C = C_0 C_1 \ldots C_{n-1}$

$C_k = \overline{a_k \oplus b_k}$

Input: a, b su 32 bit
Output: {0,1}.

Per 32 bit occorrono le seguenti porte a 2 ingressi:
- 32 XOR
- 31 AND

Si può fare di meglio sfruttando il sommatore già presente nella ALU

Operazione di uguaglianza

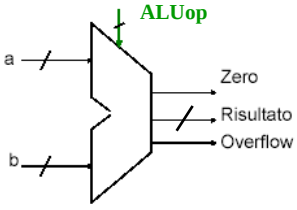
beq rs, rt label iff (rs – rt) = 0 , salta.

Occorre quindi:
- Impostare una differenza.
- Effettuare l'OR logico di tutti i bit somma.
- L'uscita dell'OR logico = 0 i due numeri sono uguali.

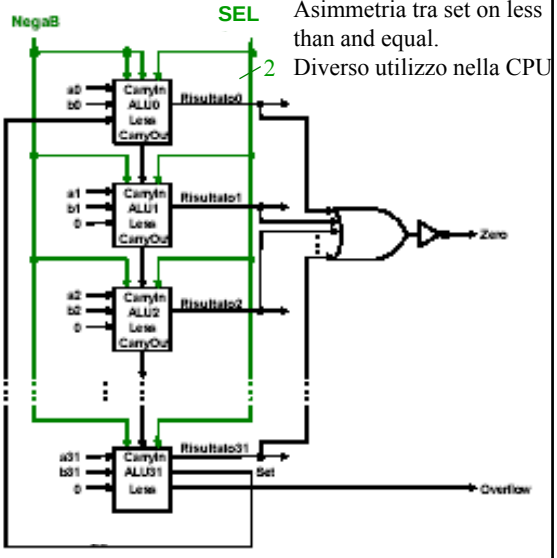
Input: a, b su 32 bit
Output: {0,1}. In questo caso viene dedicata una linea di uscita.

ALU a 32 bit: struttura finale

Operazioni possibili:
- AND
- OR
- Somma / Sottrazione
- Comparazione
- Test di uguaglianza



Sono evidenziate solamente le variabili visibili all'esterno.



Asimmetria tra set on less than and equal.
Diverso utilizzo nella CPU

Sommario

ALU ad 1 bit

ALU a 32 bit

Comparazione, Overflow, Test di uguaglianza

Tecnologie di costruzione di una ALU

Approcci tecnologici alla ALU.

Tre approcci tecnologici alla costruzione di una ALU (e di una CPU):

- Approccio strutturato. Analizzato in questa lezione.
- Approccio hardware programmabile (e.g. PAL). Ad ogni operazione corrisponde un circuito combinatorio specifico.
- Approccio ROM. E' un approccio esaustivo (tabellare). Per ogni funzione, per ogni ingresso viene memorizzata l'uscita. E' utilizzabili per funzioni molto particolari (ad esempio di una variabile). Non molto utilizzato.
- Approccio firmware (firm = stabile), o microprogrammato. Si dispone di circuiti specifici solamente per alcune operazioni elementari (tipicamente addizione e sottrazione). Le operazioni più complesse vengono sintetizzate a partire dall'algoritmo che le implementa.

Approccio ROM alla ALU

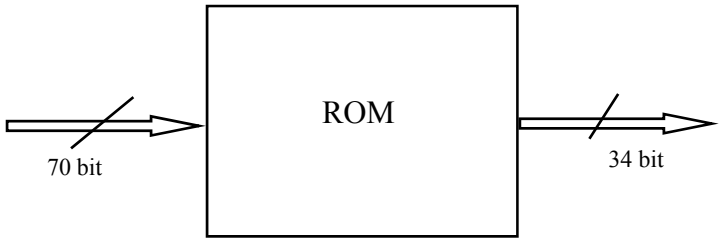
Input:
- A n bit
- B n bit
- SEL k bit
- InvertiB: 1 bit

Totale: 2*n + k + 1 bit

Output:
- S n bit
- zero 1 bit
- overflow 1 bit

Totale: n + 2 bit

Per dati su 32 bit, 5 operazioni:



Capacità della ROM: $2^{68} = 2.95..$ x 10^{20} parole di 34 bit!

(Capacità della ROM per dati su 4 bit, 5 operazioni: $2^{12} = 4{,}098$ parole di 6 bit)

L'approccio firmware

Nell'approccio firmware, viene inserita nella ALU una unità di controllo e dei registri. L'unità di controllo attiva opportunamente le unità aritmetiche ed il trasferimento da/verso i registri. Approccio *"controllore-datapath"*.

Viene inserito un microcalcolatore dentro la ALU.

Il primo microprogramma era presente nell'IBM 360 (1964).

L'approccio firmware vs hardware

La soluzione HW è più veloce ma più costosa per numero di porte e complessità dei circuiti.

La soluzione firmware risolve l'operazione complessa mediante una sequenza di operazioni semplici. E' meno veloce, ma più flessibile e, potenzialmente, adatta ad inserire nuove procedure.

La soluzione HW è percorsa per le operazioni frequenti: la velocizzazione di operazioni complesse che vengono utilizzate raramente non aumenta significativamente le prestazioni (legge di Amdahl).

Sommario

ALU ad 1 bit

ALU a 32 bit

Comparazione, Overflow, Test di uguaglianza

Tecnologie di costruzione di una ALU