Architetture dei Calcolatori

Unita' Logico-Aritmetica (ALU) e Registri

Prof. Francesco Lo Presti

Unità logico-aritmetica (ALU)

- E' la parte del processore che svolge le operazioni aritmetico-logiche
- Rete combinatoria
- Operazioni da implementare
  1. Aritmetiche
     - ✓ Somma (istruzione add)
     - ✓ Sottrazione (istruzione sub)
     - ✓ Confronto (istruzioni slt, beq e bne)
  2. Logiche
     - ✓ Operazione di AND, OR e NOR (istruzioni and, or e nor)
- Obiettivo: un circuito generale in grado di svolgere una qualunque delle operazioni sopra
  - Scelta di quale operazione eseguire in funzione di un valore specificato

Blocchi di base per costruire l' ALU

1. AND gate (c = a · b)

a	b	c = a · b
0	0	0
0	1	0
1	0	0
1	1	1

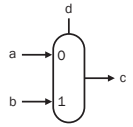
2. OR gate (c = a + b)

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)



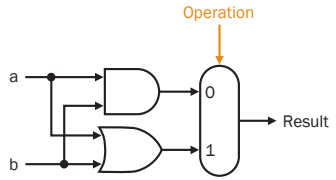
d	c
0	a
1	b

Passi per costruire l' ALU

- Implementazione incrementale dell' ALU per supportare le operazioni di
  - AND e OR logici (istruzioni and e or)
  - Addizione (istruzione add)
  - Sottrazione (istruzione sub)
  - NOR logico (istruzione nor)
  - Confronto (istruzioni slt e beq)
- Per tutti i passi (tranne uno) studieremo l' implementazione dell' ALU a 1 bit, essendo l' estensione a 32 bit di semplice realizzazione

Implementazione and e or

- Due bit di input (a, b)
- Selettore dell' operazione (multiplexer)
  - Quando vale 0, viene selezionata la linea 0 (AND)
  - Quando vale 1, viene selezionata la linea 1 (OR)

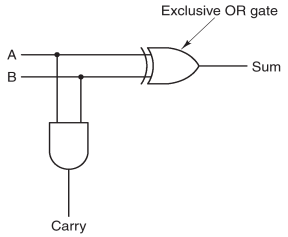


Operation=0 → Result=a∧b

Operation=1 → Result=a∨b

Semiaddizionatore (half adder)

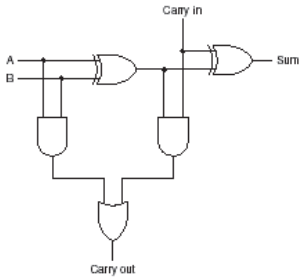
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



- Circuito logico a 2 ingressi e 2 uscite
  - Uscite: somma e riporto (*Carry*)
- Non può essere usato per la somma di numerali a più bit, dove occorre sommare anche il riporto della cifra precedente

Addizionatore completo (full adder)

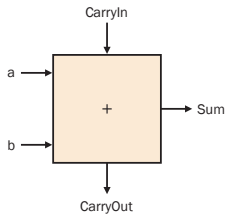
A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



- Circuito logico a 3 ingressi e 2 uscite
- Riceve il riporto dalla cifra precedente (Carry In)
- E' il generico stadio di un addizionatore su n bit
- Per implementare la somma a n-bit….
  - n-bit ripple carry adder, n-bit carry lookahead adder

Implementazione somma

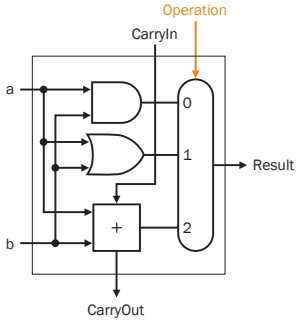
- Aggiungiamo la somma
  - Full adder ad un bit
  - Input: 2 bit da sommare (a e b) e CarryIn
  - Output: bit di somma e CarryOut



a	*b*	*CarryIn*	*Sum*	*CarryOut*
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

Implementazione somma (2)

- ALU a 1 bit che effettua and, or e somma
- Occorrono 2 bit per pilotare il multiplexer, quindi il segnale *Operation* rappresenta due linee di controllo

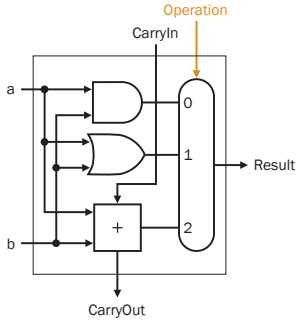


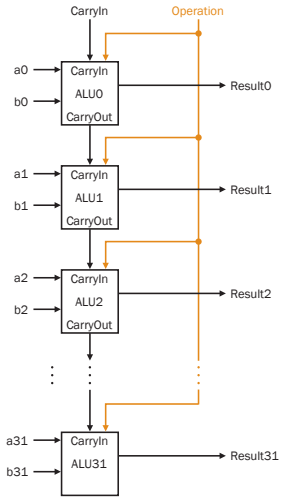
Operation=00 → Result=a∧b

Operation=01 → Result=a∨b

Operation=10 → Result=a+b

Costruzione di un ALU a 32 bit





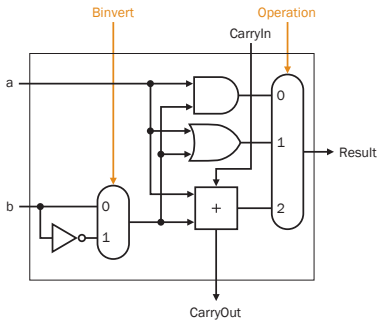
Implementazione della sottrazione

- Aggiungiamo la sottrazione
- …Sottrarre equivale a sommare il numero negativo
- Regola per negare un numero in CP2
  - Invertire i bit e sommare 1

  (a-b) = a + (-b) = a + (~b +1)
- Occorre aggiungere la possibilità di invertire i bit del secondo operando all' interno dell' ALU
- Inoltre, serve aggiungere la costante 1 per ottenere la sottrazione

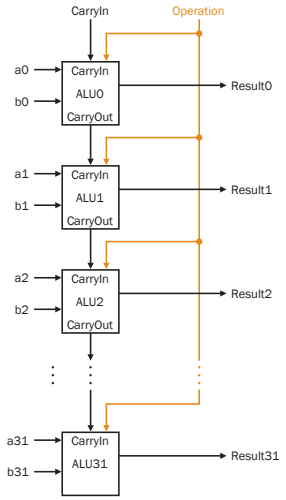
Implementazione della sottrazione (2)

- Per negare il numero
  - Invertiamo il secondo bit (secondo operando)
  - Aggiungiamo una linea di controllo (Binvert) per scegliere quale bit è da inviare all' addizionatore



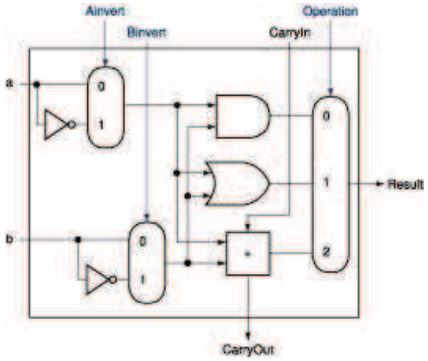
Implementazione della sottrazione (3)

- Per aggiungere la costante 1
  - Recuperiamo il riporto dell' ALU del bit meno significativo
    - ✓ Il primo CarryIn è per definizione 0
    - ✓ Se, nel caso della sottrazione, mettiamo il primo CarryIn a 1, otteniamo la somma di 1 in più
    - ✓ Questo ci permette di ottenere la sottrazione
  - Quindi
    - ✓ CarryIn = 1 se ALU_0 e si deve eseguire a-b



Implementazione nor

- Due bit di input (a, b)
- $\overline{a} + \overline{b} = \overline{a \cdot b}$
- Abbiamo già AND e NOT B
- Basta aggiungere NOT A



Operazioni di confronto

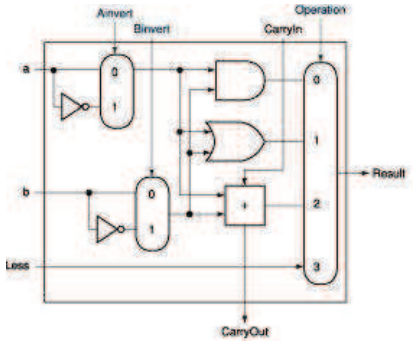
- Aggiungiamo il supporto hardware per eseguire il confronto
  - Istruzione slt (set-on-less-than)
    - ✓ slt rd, rs, rt: se rs<rt, allora rd=1
  - Istruzione beq (branch-on-equal)
    - ✓ beq rs, rt, label: se rs=rt, allora salta a label
- Soluzione: si effettua la sottrazione tra a e b
  - Se a-b è minore di 0, allora a<b (per l' istruzione slt)
    - ✓ Se a-b<0 il risultato sarà 00…01, altrimenti il risultato 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

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
  - Tutti gli altri 31 bit del risultato devono essere 0
- Aggiungiamo un segnale in ingresso *Less* aggiuntivo

Istruzione slt: ALU0-ALU30

- Soluzione per ALU1-ALU30: se dal segnale di controllo *Operation* viene selezionata la linea 3 (*Less*) di input del multiplexer, l'uscita diventa esattamente il valore di Less
  - Basta mettere 0 in ingresso a Less per ottenere il risultato voluto



ALU1-ALU30:

$CarryIn_i = CarryOut_{i-1}$

Less = 0

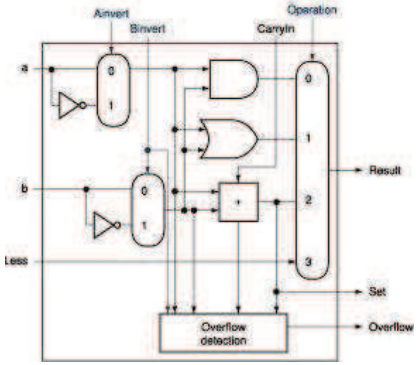
ALU0:

CarryIn = 1 (per sottrazione)

Less: vedi lucido successivo

Istruzione slt: ALU31

- Il risultato del sommatore (bit di segno) va sulla linea *Set*
  - Serve un output aggiuntivo, ossia il risultato del sommatore detto *Set*, che diventa l'input *Less* per ALU0 (bit meno significativo)
- Aggiungiamo anche una parte per individuare l'overflow

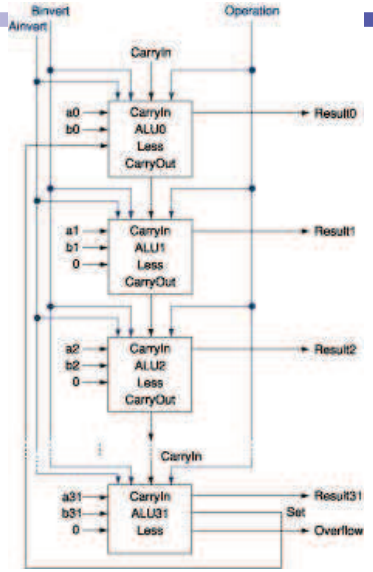


ALU31:

$CarryIn_{31} = CarryOut_{30}$

Less = 0

ALU a 32 bit con istruzione slt

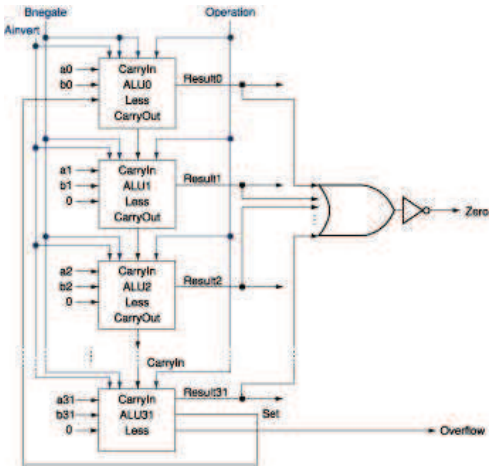


Ottimizzazione delle linee di controllo

- Nella sottrazione, Binvert e CarryIn di ALU0 sono entrambi pari a 1
- Nelle operazioni logiche (tranne nor) e nella somma, Binvert e CarryIn di ALU0 sono entrambi pari a 0
- Quindi, per ottimizzare il controllo, fondiamo le due linee Binvert e CarryIn di ALU0 in un'unica linea chiamata Bnegate

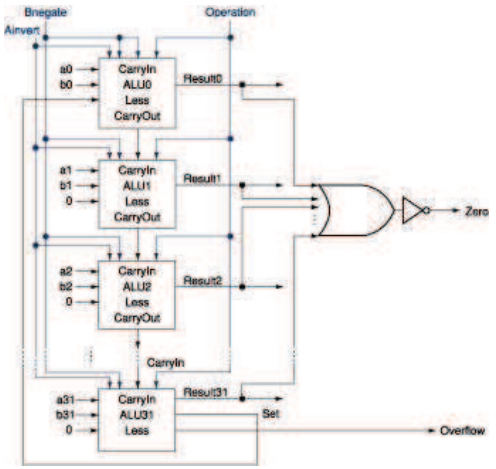
ALU a 32 bit con istruzione beq

- Soluzione:
  - Mettere tutte le linee dei risultati in OR
  - Se il risultato è zero, allora la linea *Zero* viene messa a 1
  - Se a=b, allora la linea *Zero* è 1



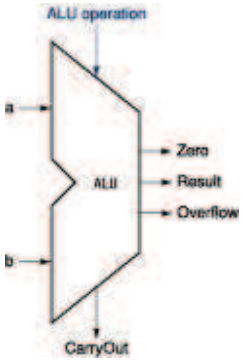
ALU a 32 bit: Linee di Controllo

- Interpretazione delle linee di controllo
- Ainvert (1 bit),
- Bnegate (1 bit) e
- Operation (2 bit)
  - 0000 = and
  - 0001 = or
  - 0010 = add
  - 0110 = sub
  - 0111 = slt
  - 1100 = nor



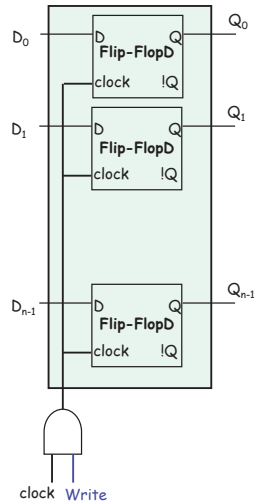
ALU a 32 bit

- Simbolo tipicamente usato per rappresentare l' ALU
  - Questo simbolo è anche usato per rappresentare un addizionatore, quindi è etichettato con ALU o adder a seconda del caso



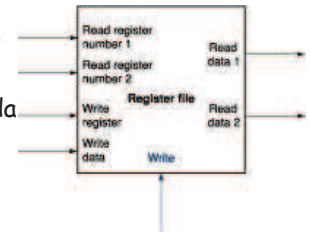
Registro

- Registro
  - Simile ad un flip flop D eccetto
    - ✓ n bit di ingresso e uscita
    - ✓ Input Write
  - Write:
    - ✓ Se negato (0): i dati in uscita (Data Out) non cambiano
    - ✓ Se affermato (1): i dati in uscita (Data Out) divengono uguali ai dati in ingresso (Data In)



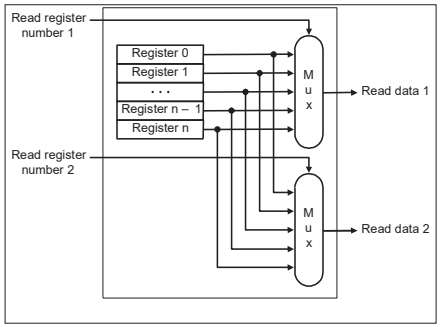
Banco di registri (register file)

- Banco di registri ad accesso rapido per memorizzare temporaneamente gli operandi usati nelle istruzioni
- Nel MIPS il banco dei registri è composto da 32 registri generali
  - Due porte in lettura da 32 bit
    - ✓ Read data 1/2
  - Una porta in scrittura da 32 bit
    - ✓ Write data
  - Tre porte per selezionare i registri da 5 bit
    - ✓ Read register #1 (#2) : primo (secondo) registro da leggere
    - ✓ Write register: registro da scrivere
- Write: segnale di controllo
  - In AND con il clock (non mostrato)
  - Solo se Write=1 il valore in Write data e' scritto nel registro indicato da Write Reg.



Banco di registri (2)

- Un banco di registri può essere implementato con un multiplexer per ciascuna porta read, un decoder per ciascuna porta write ed un array di registri costruiti partendo da flip-flop D
- Esempio: implementazione di due porte read per un banco di registri composto da *n* registri



Banco di registri (3)

- Esempio: implementazione di una porta write per un banco di registri composto da *n* registri