



# La CPU a singolo ciclo

Prof. Alberto Borghese  
Dipartimento di Scienze dell'Informazione  
[borgnese@dsi.unimi.it](mailto:borgnese@dsi.unimi.it)

Università degli Studi di Milano

Riferimento sul Patterson: capitolo 5 (fino a 5.4)



## Sommario

**L'Architettura di Von Neuman, architetture CISC e RISC.**

La CPU.

Costruzione di una CPU per le istruzioni di tipo R

Costruzione di una CPU per le istruzioni di tipo I (memoria).

Costruzione di una CPU per le istruzioni di tipo I (salti).

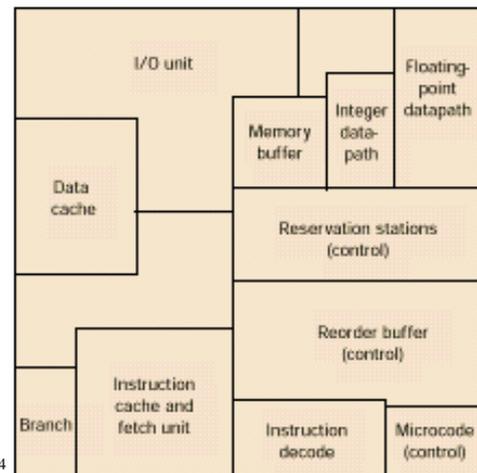


## Architettura di riferimento degli elaboratori

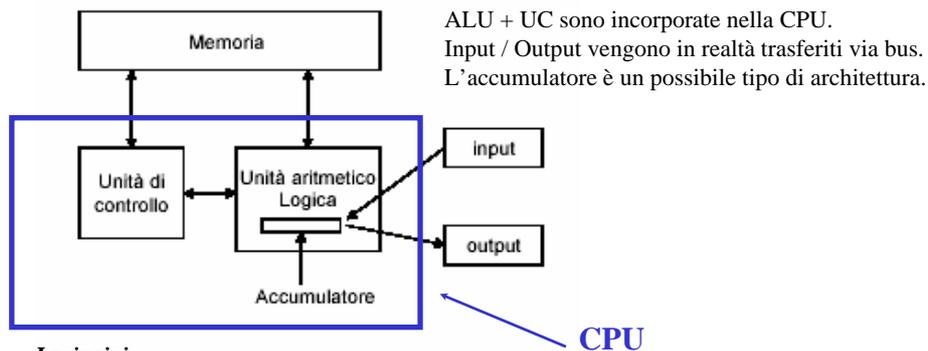


- Elementi principali di un elaboratore:
  - Unità centrale di elaborazione (*Central Processing Unit - CPU*)
  - Memoria di lavoro o memoria principale (*Main Memory - MM*)

- Pentium Pro. 5,5 Milioni di transistor su 306mm<sup>2</sup> (2cm x 1.5cm) con cache esterna da 31 milioni.



## Architettura di Von Neumann



### I principi:

- I dati e le istruzioni sono memorizzate in una memoria read/write.
- Il contenuto della memoria può essere recuperato in base alla sua posizione, e non è funzione del tipo di dato.
- L'esecuzione procede sequenzialmente da un'istruzione alla seguente.
- Già' vista e modificata



## Alcuni tipi di architetture



Accumulator (1 register = 1 indirizzo di memoria).

1 address	add A	$\text{acc} \leftarrow \text{acc} + \text{mem}[A]$
1+x address	addx A	$\text{acc} \leftarrow \text{acc} + \text{mem}[A + x]$

Stack (posso operare solo sui dati in cima allo stack):

0 address	add	$\text{tos} \leftarrow \text{tos} + \text{next}$
-----------	-----	--------------------------------------------------

General Purpose Register (tanti diversi indirizzi di memoria quanti sono i registri, indirizzamento indiretto):

2 address	add A B	$\text{EA}(A) \leftarrow \text{EA}(A) + \text{EA}(B)$
3 address	add A B C	$\text{EA}(A) \leftarrow \text{EA}(B) + \text{EA}(C)$

Load/Store (posso operare solamente sui dati contenuti nei registri. Devo prima caricarli dalla memoria).

3 address	add Ra Rb Rc	$\text{Ra} \leftarrow \text{Rb} + \text{Rc}$
	load Ra Rb	$\text{Ra} \leftarrow \text{mem}[\text{Rb}]$
	store Ra Rb	$\text{mem}[\text{Rb}] \leftarrow \text{Ra}$



## Architetture LOAD/STORE



- Il numero dei registri ad uso generale (ad esempio 32 registri da 32 bit ciascuno) non è sufficientemente grande da consentire di memorizzare tutte le variabili di un programma  $\Rightarrow$  ad ogni variabile viene assegnata una locazione di memoria nella quale trasferire il contenuto del registro quando questo deve essere utilizzato per contenere un'altra variabile.
- *Architetture LOAD/STORE*: gli operandi dell'ALU possono provenire soltanto dai registri ad uso generale contenuti nella CPU e **non** possono provenire dalla memoria. Sono necessarie apposite istruzioni di:
  - *caricamento (LOAD)* dei dati da memoria ai registri;
  - *memorizzazione (STORE)* dei dati dai registri alla memoria.

Vedremo quando parleremo di memoria in che modo questa architettura può essere particolarmente efficiente.



## Utilizzo architettura Intel 80x86: le 10 istruzioni più frequenti



° Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	<b>Total</b>	<b>96%</b>

° Simple instructions dominate instruction frequency => RISC



## Architetture di tipo RISC (*Reduced Instruction Set Computer*)



- Ispirate al principio di eseguire soltanto istruzioni semplici: le operazioni complesse vengono scomposte in una serie di istruzioni più semplici da eseguire in un ciclo base ridotto, con l'obiettivo di migliorare le prestazioni ottenibili dalle *CPU CISC*.
- Caratterizzate da istruzioni molto semplificate.
- Gli operandi dell'*ALU* possono provenire dai registri ma *non* dalla memoria. Per il trasferimento dei dati da memoria ai registri e viceversa si utilizzano delle apposite operazioni di caricamento (*load*) e di memorizzazione (*store*)  
⇒ *architetture load/store*.



## CPU di tipo RISC (*Reduced Instruction Set Computer*)



- CPU relativamente semplice  $\Rightarrow$  si riducono i tempi di esecuzione delle singole istruzioni, che sono però meno potenti delle istruzioni CISC.
- Dimensione *fissa* delle istruzioni  $\Rightarrow$  più semplice la gestione della fase di prelievo (*fetch*) e della codifica delle istruzioni da eseguire.



## CPU di tipo CISC (*Complex Instruction Set Computer*)



- Caratterizzate da elevata complessità delle istruzioni eseguibili ed elevato numero di istruzioni che costituiscono l'insieme delle istruzioni.
- Numerose modalità di indirizzamento per gli **operandi** dell'ALU che possono provenire da registri oppure da memoria, nel qual caso l'indirizzamento può essere diretto, indiretto, con registro base, ecc.



## CPU di tipo CISC (Complex Instruction Set Computer)



- Dimensione *variabile* delle istruzioni a seconda della modalità di indirizzamento di ogni operando  $\Rightarrow$  complessità di gestione della fase di prelievo o *fetch* in quanto a priori non è nota la lunghezza dell'istruzione da caricare.
- Elevata complessità della *CPU* stessa (cioè dell'hardware relativo) in termini degli elementi che la compongono con la conseguenza di rallentare i tempi di esecuzione delle operazioni. Elevata profondità dell'albero delle porte logiche, utilizzato per la decodifica.



## I diversi formati di istruzioni



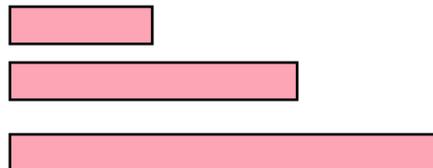
Variabile



Fisso  
(MIPS)



Ibrido



Il formato fisso consente di massimizzare la velocità, il formato ibrido consente di minimizzare la lunghezza del codice.



## Sommario



L'Architettura di Von Neuman, architetture CISC e RISC.

### La CPU.

Costruzione di una CPU per le istruzioni di tipo R

Costruzione di una CPU per le istruzioni di tipo I (memoria).

Costruzione di una CPU per le istruzioni di tipo I (salti).



## Obiettivo

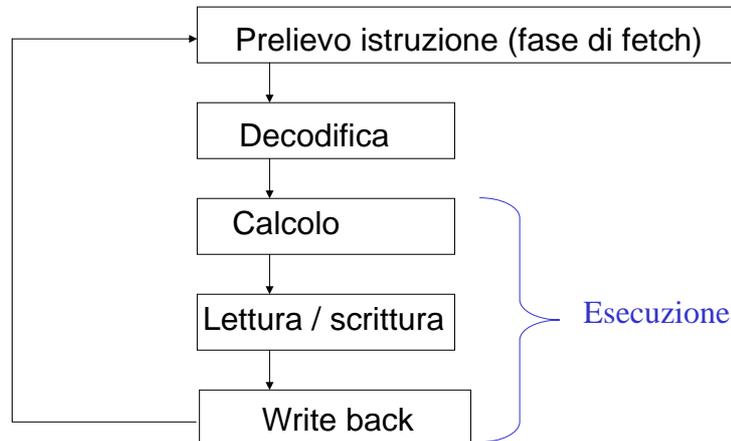


Costruzione di una CPU completa che sia in grado di eseguire:

- Accesso alla memoria in lettura (lw) o scrittura (sw).
- Istruzioni logico-matematiche (e.g. add, sub, and....).
- Istruzioni di salto condizionato (branch) o incondizionato (jump).



## Ciclo di esecuzione di un'istruzione MIPS



## I componenti di un'architettura

### CPU

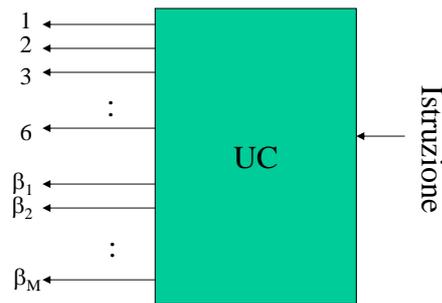
- Banco di registri (*Register File*) ad accesso rapido, in cui memorizzare i dati di utilizzo più frequente. Il tempo di accesso ai registri è circa 10 volte più veloce del tempo di accesso alla memoria principale.
- Registro *Program counter (PC)*. Contiene l'indirizzo dell'istruzione corrente da aggiornare durante l'evoluzione del programma, in modo da prelevare dalla memoria la corretta sequenza di istruzione;
- Registro *Instruction Register (IR)*. Contiene l'istruzione in corso di esecuzione. Questo registro verrà utilizzato più avanti nelle architetture multi-ciclo.
- Unità per l'esecuzione delle operazioni aritmetico-logiche (*Arithmetic Logic Unit - ALU*). I dati forniti all'*ALU* possono provenire da registri oppure direttamente dalla memoria, a seconda delle modalità di indirizzamento previste;
- Unità aggiuntive per elaborazioni particolari come unità aritmetiche per dati in virgola mobile (*Floating Point Unit - FPU*), sommatore ausiliari, ecc.;
- **Unità di controllo**. Controlla il flusso e determina le operazioni di ciascun blocco.

### MEMORIA PRINCIPALE



## L'unità di controllo

- Unità di controllo coordina i flussi di informazione (è il “cervello” della CPU):
- 1) abilitando le vie di comunicazione opportune a seconda dell'istruzione in corso di esecuzione.
- 2) selezionando l'operazione opportuna delle ALU.



## Come funziona una UC

- Usa un registro, il Program Counter (PC) per ottenere l'indirizzo dell'istruzione.
- Preleva l'istruzione dalla memoria e la inserisce nell'IR.
- Capisce di che tipo di istruzione si tratta (decodifica).
  - usa l'istruzione stessa per decidere cosa fare esattamente.
- Legge il contenuto dei registri.

*Da qui le istruzioni si differenziano.*

- Calcolo: utilizzo dell'ALU dopo aver letto i registri:
  - per calcolare l'indirizzo in memoria.
  - per eseguire un'operazione logico-aritmetica.
  - per effettuare il test.
- Accesso alla memoria.
- Scrittura del risultato nel register file.



## Rapporto UC - Dati



La CPU è un'architettura del tipo: Controllore - Data-path

Dal mondo esterno



*Fase comune nel ciclo di esecuzione:*

- Fase di fetch
- Decodifica (generazione dei segnali di controllo)

*Fase diversa:* Esecuzione (Calcolo, Accesso memoria, WriteBack)



## Sommario



L'Architettura di Von Neuman, architetture CISC e RISC.

La CPU.

**Costruzione di una CPU per le istruzioni di tipo R**

Costruzione di una CPU per le istruzioni di tipo I (memoria).

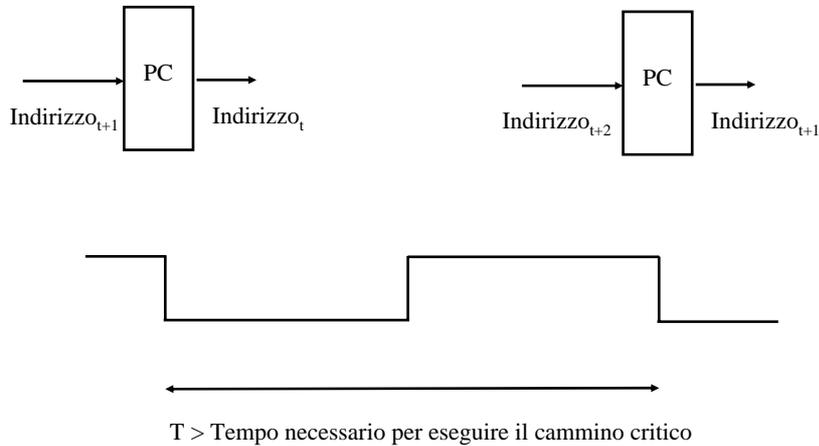
Costruzione di una CPU per le istruzioni di tipo I (salti).



## Temporizzazione



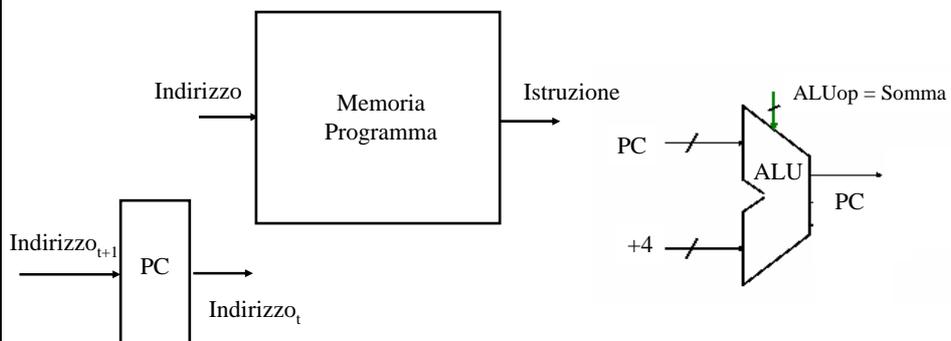
1 istruzione per ciclo di clock. Temporizzazione del PC.



## Fase di fetch

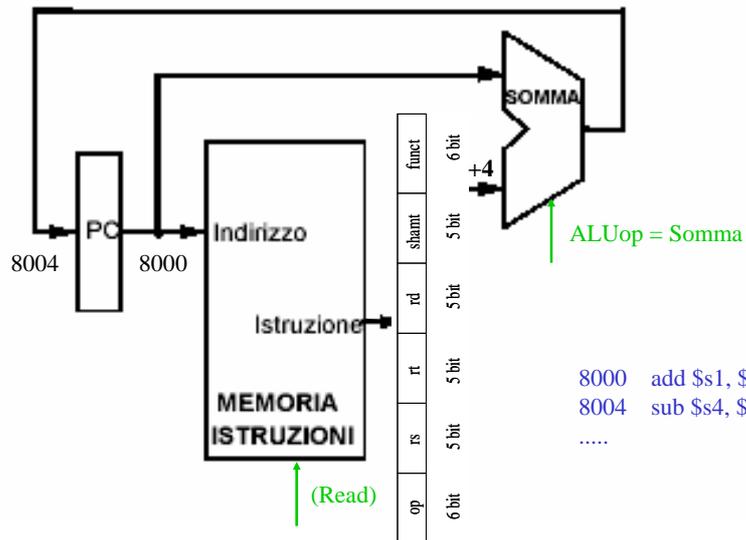


- 1) Memorizzare l'indirizzo dell'istruzione nel PC.
- 2) Leggere l'istruzione dalla memoria.
- 3) Aggiornare l'indirizzo in modo che in PC sia contenuto l'indirizzo dell'istruzione successiva.





## Circuito della fase di fetch



## Istruzioni di tipo R



R	op	rs	rt	rd	shamt	funct
	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

add \$s1, \$s2, \$s3




## Fase di decodifica

- 1) Leggo l'istruzione e genero i segnali di controllo opportuni.
- 2) Leggo il contenuto dei registri.

funct

shamt

rd

rt

rs

op

6 bit

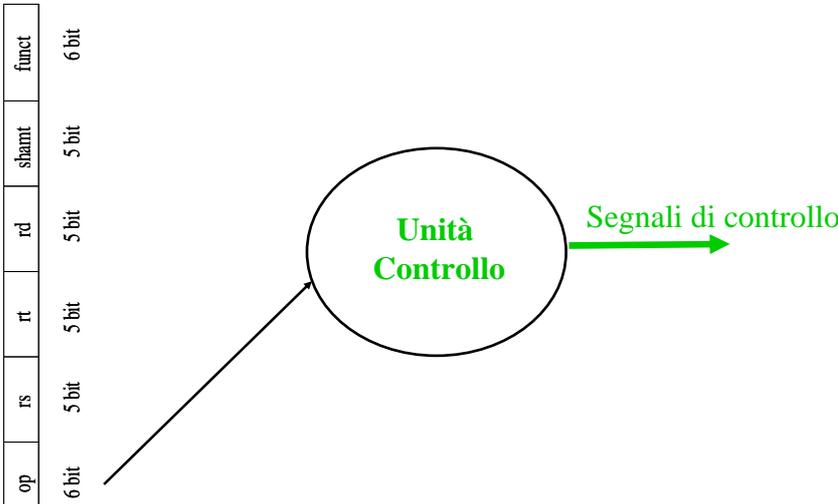
5 bit

5 bit

5 bit

5 bit

6 bit



A.A. 2006-21

25/44

<http://homes.dsi.unimi.it/~borgnese>




## Register file

Banco di registri utilizzabile come memoria  
Può essere scritto o letto.

#Reg read 1

#Reg read 2

#Reg write

Contenuto Write

Insieme di 32 registri da 32 bit

R W

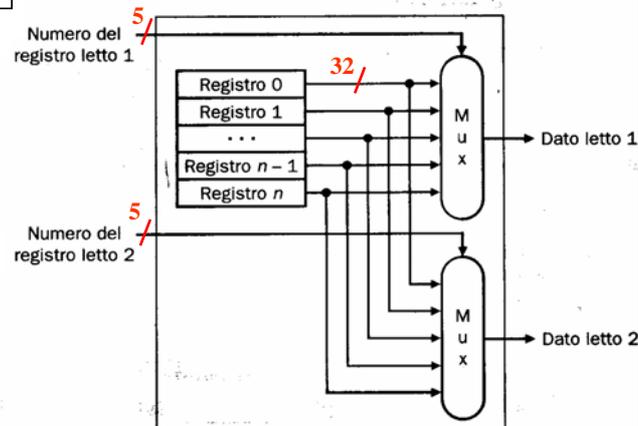
Contenuto 1

Contenuto 2

Numero del registro letto 1

Numero del registro letto 2

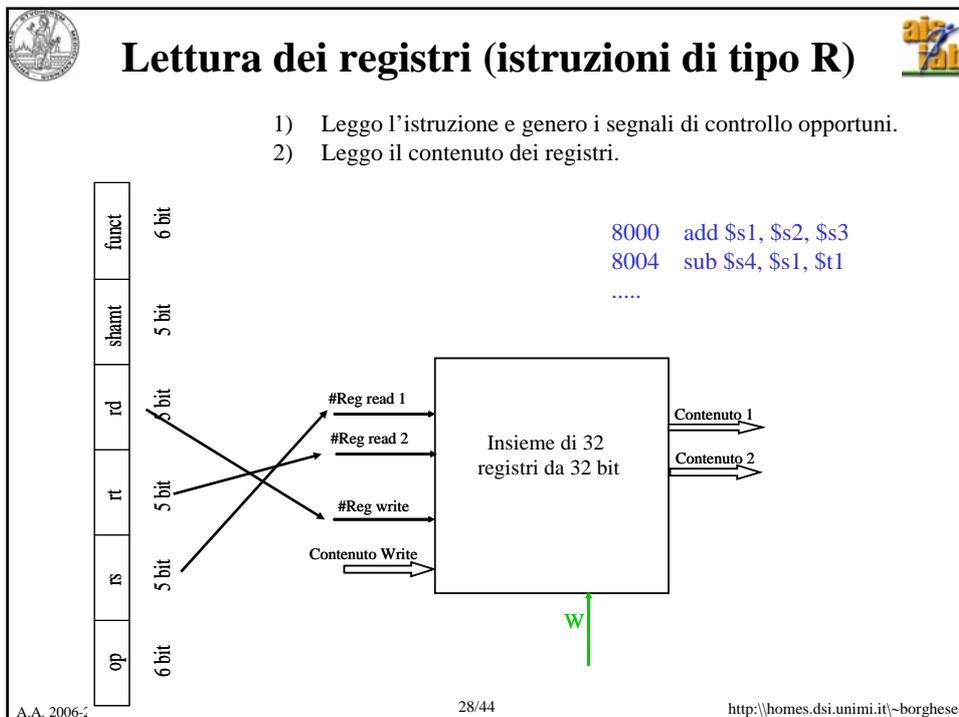
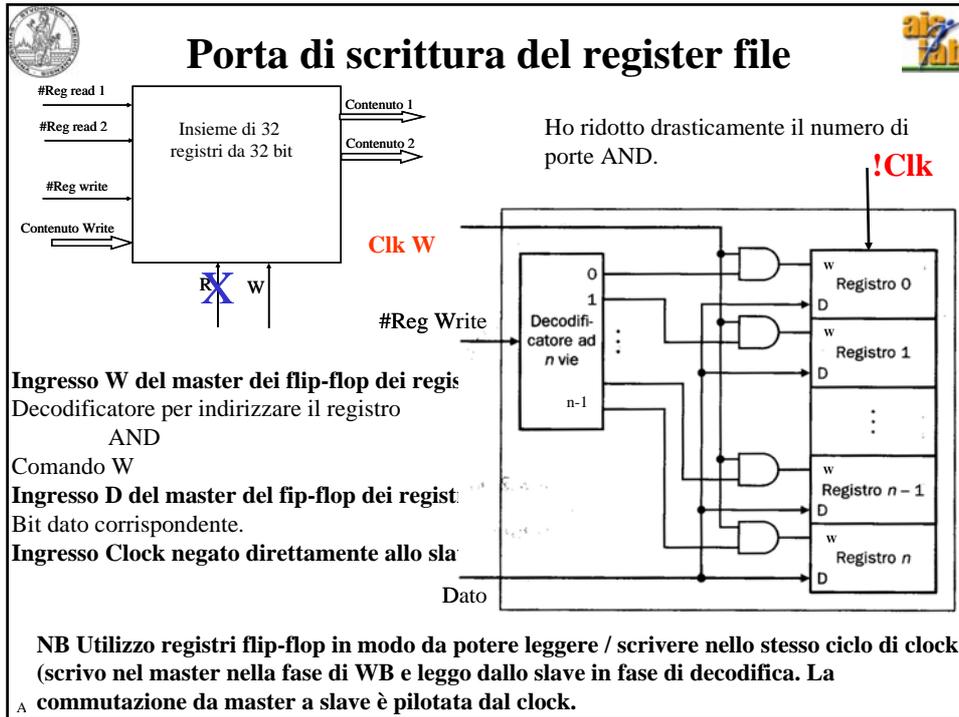
Un mux per ogni porta di lettura.



A.A. 2006-2007

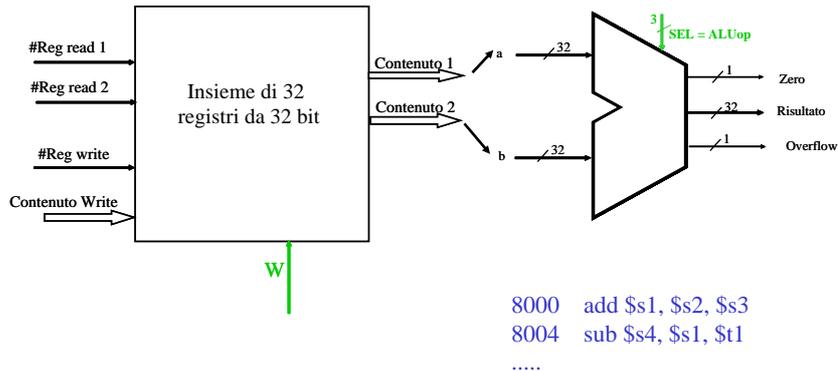
26/44

<http://homes.dsi.unimi.it/~borgnese>

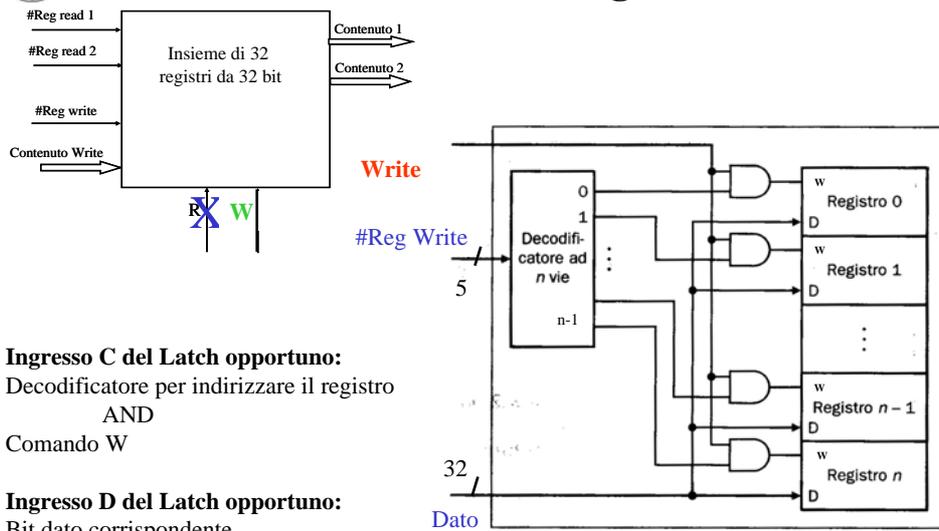


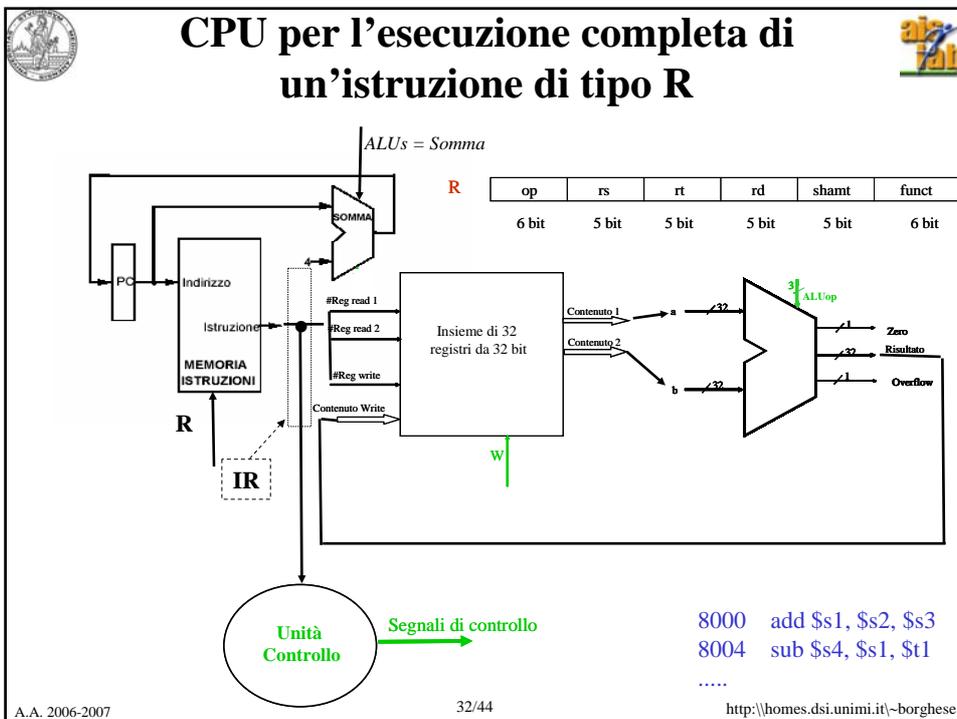
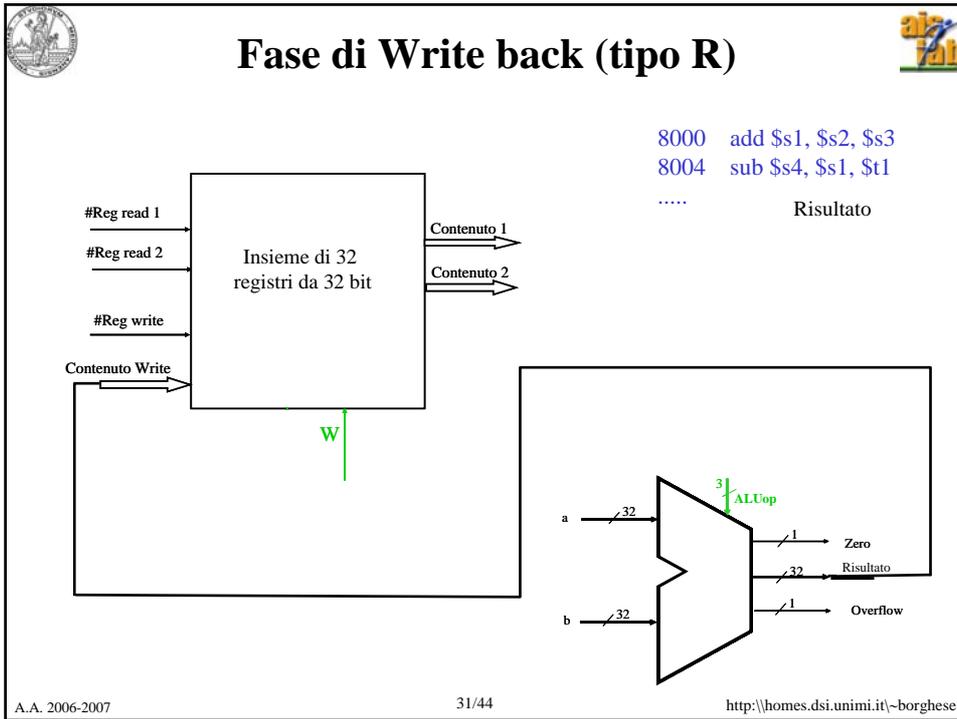


## Fase di Calcolo (tipo R)



## Porta di scrittura del register file







## Sommario



L'Architettura di Von Neuman, architetture CISC e RISC.

La CPU

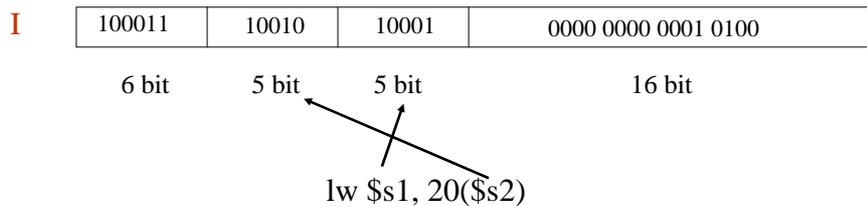
Costruzione di una CPU per le istruzioni di tipo R

Costruzione di una CPU per le istruzioni di tipo I (memoria).

Costruzione di una CPU per le istruzioni di tipo I (salti).



## Istruzioni di tipo I: lw/sw



L'indirizzo di memoria sarà:

Base <\$s2> 0100 1000 0011 0001 1011 1011 1011 1011 +  
 Offset 0000 0000 0001 0100 +

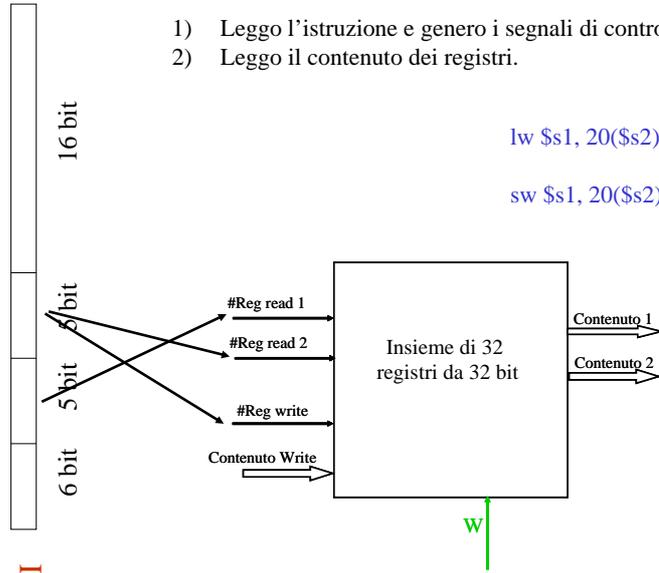
Indirizzo dato 0100 1000 0011 0001 1011 1011 1100 1111



## Lettura dei registri (istruzioni di tipo I)



- 1) Leggo l'istruzione e genero i segnali di controllo opportuni.
- 2) Leggo il contenuto dei registri.

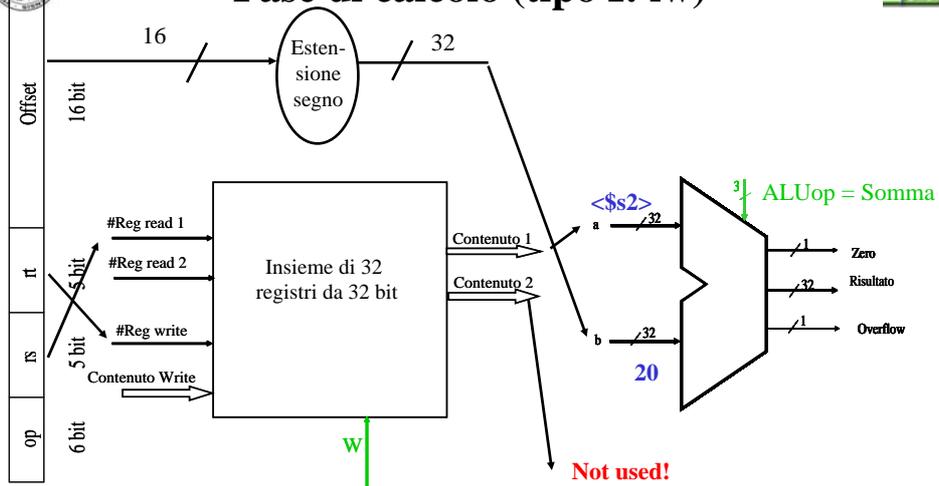


lw \$s1, 20(\$s2)

sw \$s1, 20(\$s2)

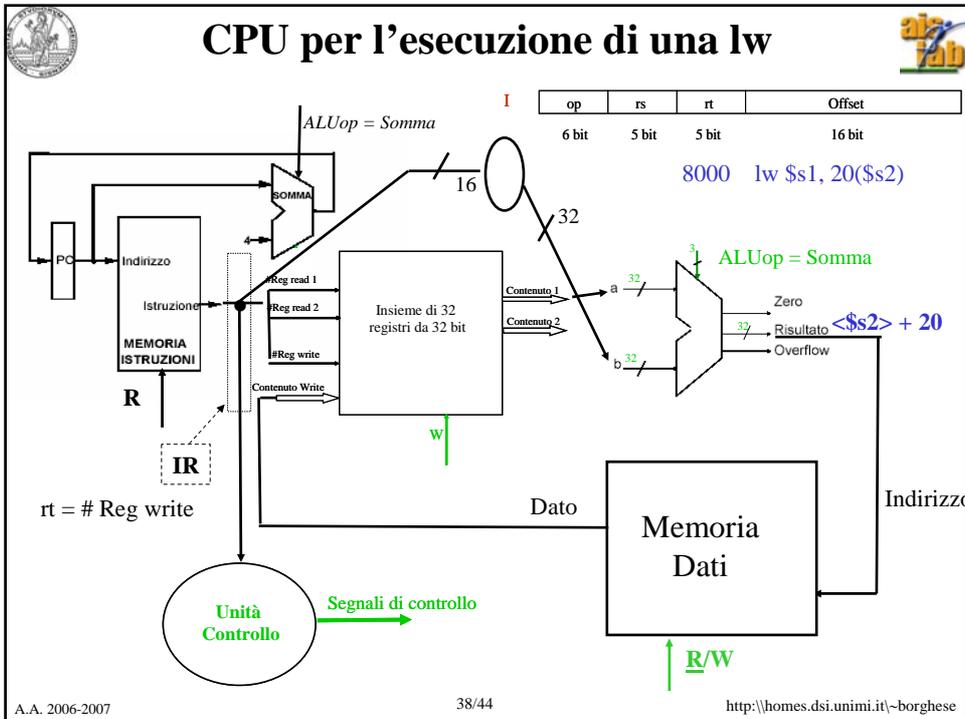
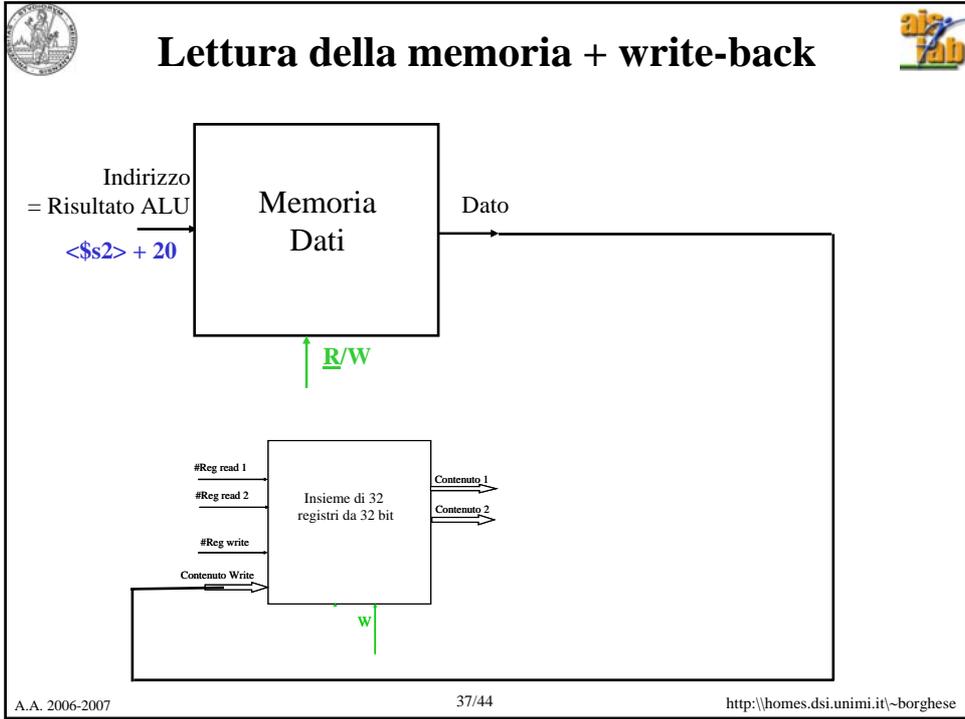


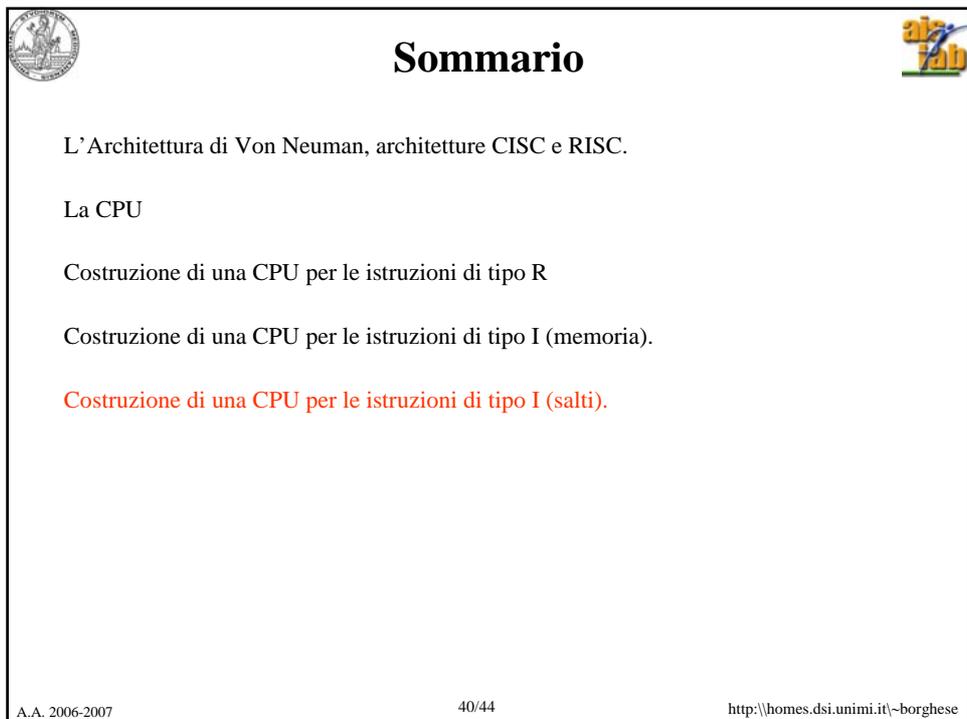
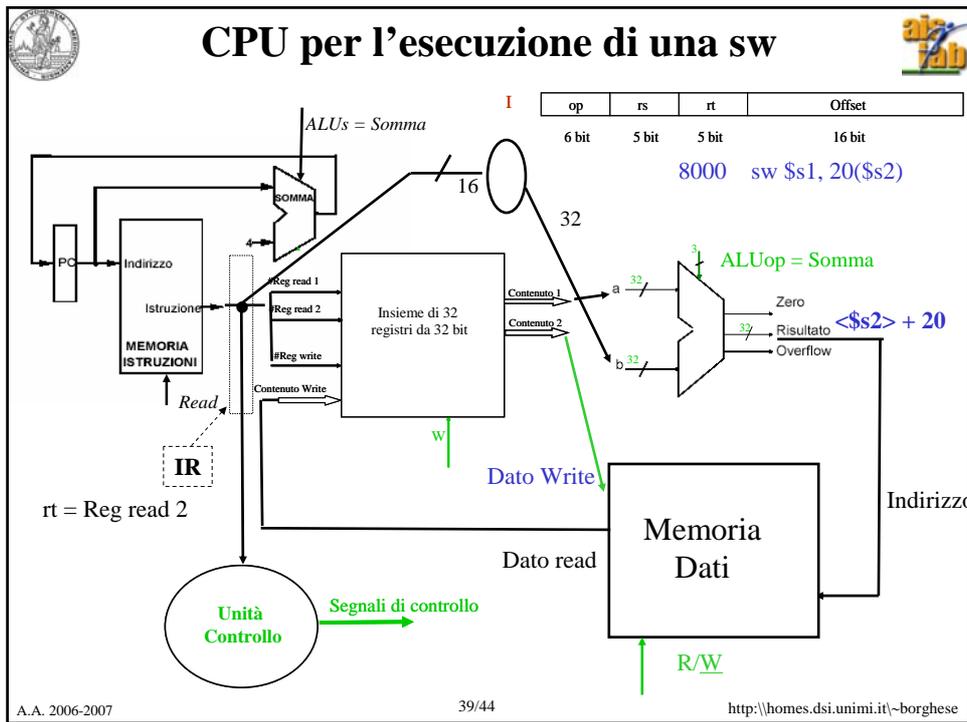
## Fase di calcolo (tipo I: lw)



Il Risultato è un indirizzo della memoria

8000 lw \$s1, 20(\$s2)







## Istruzioni di tipo I: branch



000100	10001	10010	0000 0000 0000 0101
--------	-------	-------	---------------------

beq \$s1, \$s2, 20

L'indirizzo di salto sarà determinato in due passi:

- Calcolo dello spiazzamento: Offset \* 4.
- Deteminazione dell'indirizzo di salto come:

Base (PC)	0100 1000 0011 0001	1011 1011 1011 10 11	+
Offset		00 0000 0000 0001 01 (00)	+
Indirizzo salto	0100 1000 0011 0001	1011 1011 1100 10 11	

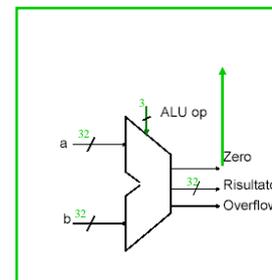
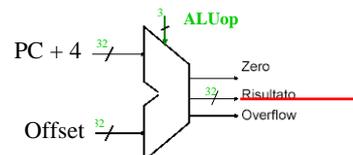
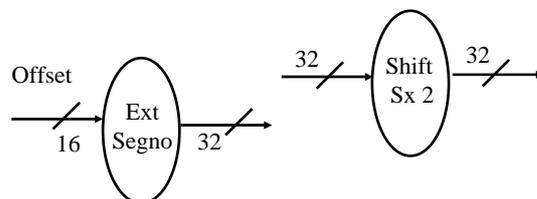
- Determinare se l'uguaglianza è vera.
- Calcolare l'indirizzo di salto.



## Fase di calcolo della beq



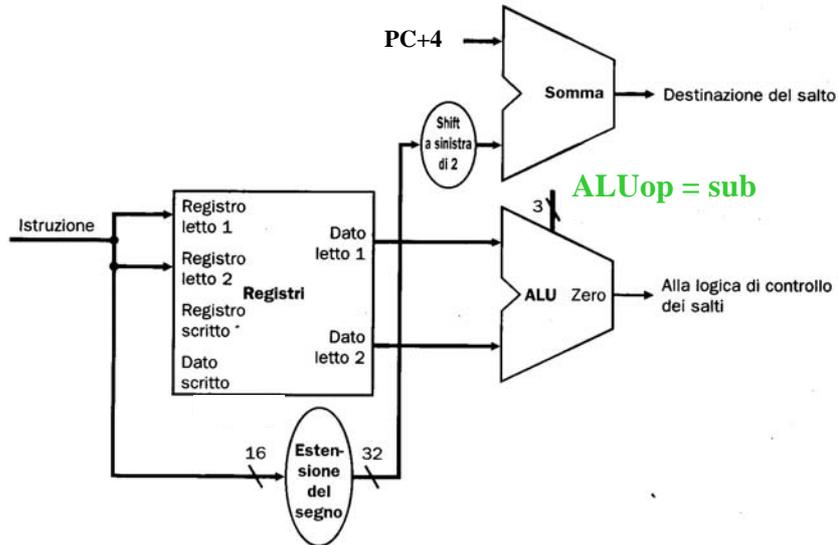
- Estensione dell'offset su 32 bit.
- Moltiplicazione per 4 dell'offset.
- Somma del PC con l'estensione del segno.
- Controllo se il contenuto dei registri è uguale.



000100	10001	10010	0000 0000 0000 0101
Cod Op	rs	rt	Offset



## Circuito di calcolo della beq



## Sommario



L'Architettura di Von Neuman, architetture CISC e RISC.

La CPU

Costruzione di una CPU per le istruzioni di tipo R

Costruzione di una CPU per le istruzioni di tipo I (memoria).

Costruzione di una CPU per le istruzioni di tipo I (salti).