


# Architettura degli elaboratori - II
# CPU a ciclo singolo

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

Università degli Studi di Milano

Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.




## Sommario

La CPU

Costruzione di una CPU per le istruzioni di tipo R

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

# Obbiettivo di un'architettura

Elabora in modo adeguato un input per produrre l'output.

- Le unità di *ingresso* (tastiera, mouse, rete, interfacce con dispositivi di acquisizione, ecc.) permettono al calcolatore di acquisire informazioni dall'ambiente esterno.
- L'architettura di elaborazione.

- Le unità di *uscita* (terminale grafico, stampanti, rete, ecc.) consentono al calcolatore di comunicare i risultati ottenuti dall'elaborazione all'ambiente esterno.



# Cosa fa un elaboratore?

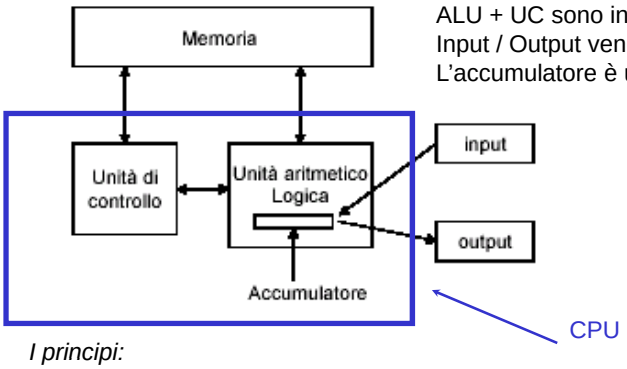
- Algoritmi (sequenza di istruzioni).
  Calcoli (calcolatore).
  **Operazioni logiche** (elaboratore).
- Programma (Ada Lovelace, 1830) = *Algoritmi in Software.*

**<u>Come lo fa?</u>** *Hardware*.

Input ==> Elaborazione ==> Output

- Terza rivoluzione della nostra civiltà: la rivoluzione agricola, la rivoluzione industriale e la rivoluzione dell'informatica.

# Architettura di Von Neumann




ALU + UC sono incorporate nella CPU.
Input / Output vengono in realtà trasferiti via bus
L'accumulatore è un possibile tipo di architettura

*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.

•Gia' vista e modificata (evoluzione nel tempo).



# Alcuni tipi di architetture

Accumulator (1 register = 1 indirizzo di memoria).

| | | |
|---|---|---|
| 1 address | add A | acc ← acc + mem[A] |
| 1+x address | addx A | acc ← acc + mem[A + x] |

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

| | | |
|---|---|---|
| 0 address | add | tos ← tos + next |

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

| | | |
|---|---|---|
| 2 address | add A B | EA(A) ← EA(A) + EA(B) |
| 3 address | add A B C | EA(A) ← EA(B) + EA(C) |

Indirizzamento misto (registro, stack, ….)

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

| | | |
|---|---|---|
| 3 address | add Ra Rb Rc | Ra ← Rb + Rc |
| | load Ra Rb | Ra ← mem[Rb] |
| | store Ra Rb | mem[Rb] ← 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 ⇒ 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.



# 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.

- Dimensione *variabile* delle istruzioni a seconda della modalità di indirizzamento di ogni operando ⇒ 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.

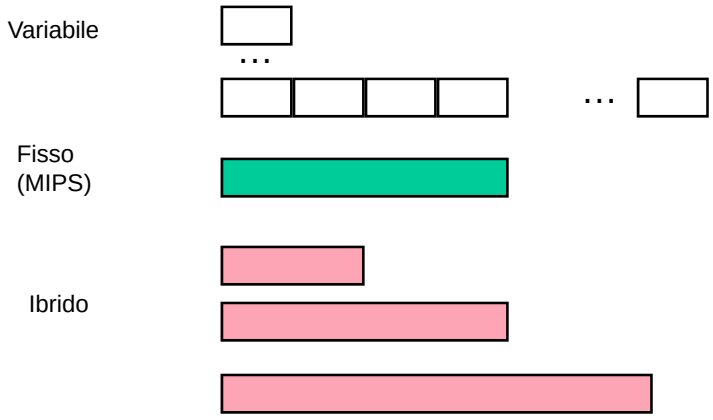
# Utilizzo architettura Intel 80x86: le 10 istruzioni più frequenti

| 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% |
| | Total | 96% |

- Simple instructions dominate instruction frequency => RISC



# I diversi formati di istruzioni




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

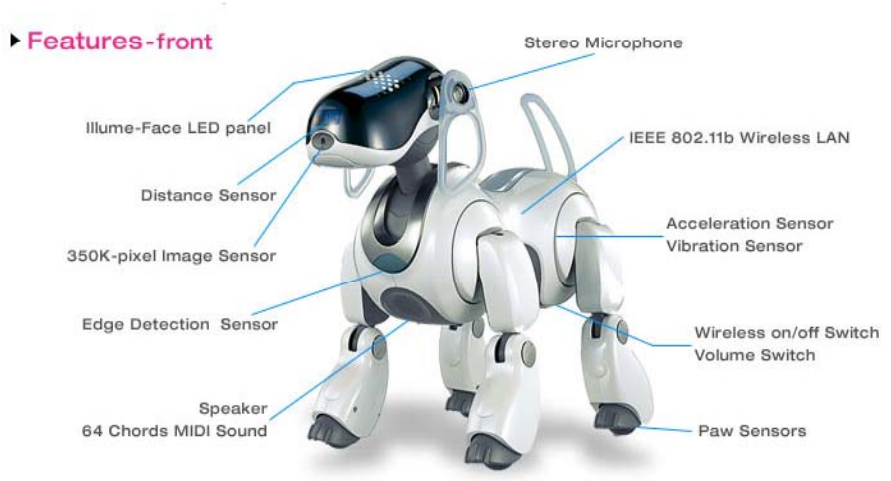
# Architetture di tipo RISC (*Reduced Instruction Set Computer*)

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



# Architettura base del corso – MIPS
## MIPS Technologies

AIBO (Sony, 2003) – MIPS 7000, sistemi embedded che montano Windows CE, PlayStation 2, router, gateway...

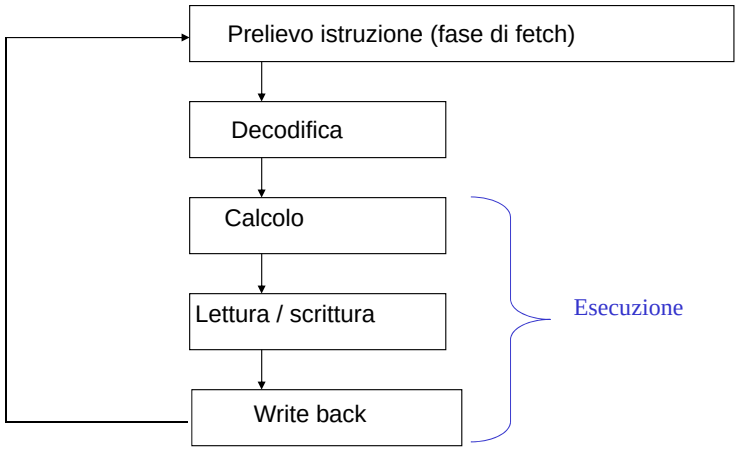


# Architettura MIPS

- Architettura MIPS appartiene alla famiglia delle architetture **RISC (Reduced Instruction Set Computer)** sviluppate dal 1980 in poi
  - Esempi: Sun Sparc, HP PA-RISC, IBM Power PC, DEC Alpha, Silicon Graphics, AIBO-Sony, ARM.
- Principali obiettivi delle architetture RISC:
  - Semplificare la progettazione dell'hardware e del compilatore
  - Massimizzare le prestazioni
  - Minimizzare i costi



# 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)*, sommatori ausiliari, ecc.;
- ***Unità di controllo***. Controlla il flusso e determina le operazioni di ciascun blocco.

**MEMORIA PRINCIPALE**



# Codifica delle istruzioni

- Tutte le istruzioni MIPS hanno la **stessa** dimensione **(32 bit) – Architettura RISC.**
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
  - il tipo di istruzione è riconosciuto in base al valore di alcuni bit **(6 bit)** più significativi **(codice operativo - *OPCODE*)**
- Le istruzioni MIPS sono di **3** tipi (formati):
  - **Tipo R (register) – Lavorano su 3 registri.**
    - Istruzioni aritmetico-logiche.
  - **Tipo I (immediate) – Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
    - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
  - **Tipo J (jump) – Lavora senza registri: codice operativo + indirizzo di salto.**
    - Istruzioni di salto incondizionato.

| | 6-bit | 5-bit | 5-bit | 5-bit | 5-bit | 6-bit |
|---|---|---|---|---|---|---|
| R | op | rs | rt | rd | shamt | funct |
| I | op | rs | rt | indirizzo | | |
| J | op | indirizzo | | | | |

# Istruzioni

| | | | | | | |
|---|---|---|---|---|---|---|
| `add $s1, $s2, $s3` | 000000 | 10010 | 10011 | 10001 | 00000 | 100000 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| `beq $s1, $s2, -100` | 000100 | 10001 | 10010 | 1111 | 1111 | 1110 | 0111 |
| `lw $t0, 32 ($s3)` | 100011 | 10011 | 01000 | 0000 | 0000 | 0010 | 0000 |
| `sw $t0, 32 ($s3)` | 101011 | 10011 | 01000 | 0000 | 0000 | 0010 | 0000 |
| `addi $t0, $s3, 64` | 001000 | 10011 | 01000 | 0000 | 0000 | 0100 | 0000 |
| `j 0x80000` | 000010 | 00 0000 0100 0000 0000 0000 0000 | | | | | |



# Sommario

La CPU

Costruzione di una CPU per le istruzioni di tipo R

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

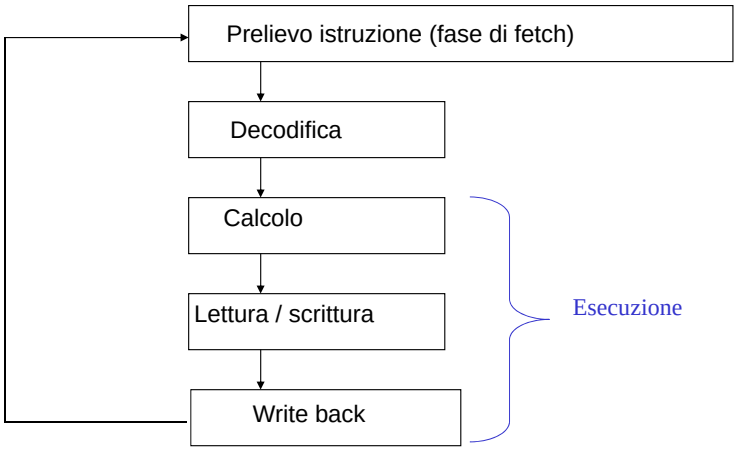
# Obbiettivo

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



# Come funziona una CPU?

- 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 test (uguaglianza, disuguaglianza, <…).

- Accesso alla memoria.

- Scrittura del risultato nel register file.



# Come funziona una CPU?

- 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.*

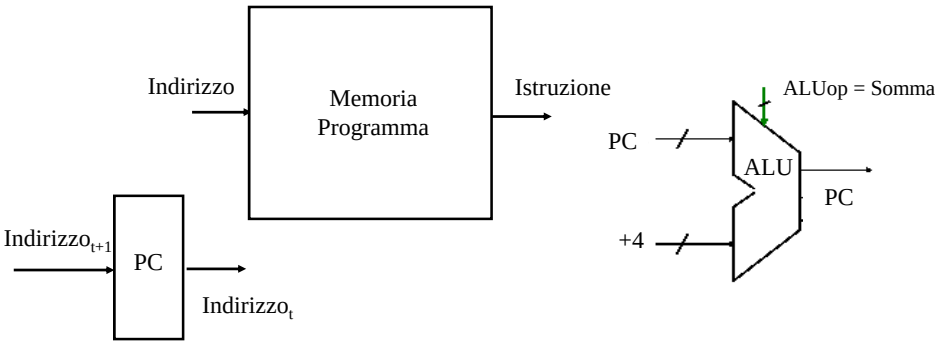
- Calcolo: utilizzo dell'ALU dopo aver letto i registri:
  - per calcolare l'indirizzo in memoria.
  - per eseguire un'operazione logico-aritmetica.
  - per effettuare test (uguaglianza, disuguaglianza, <…).

- Accesso alla memoria.

- Scrittura del risultato nel register file.

# Fase di fetch

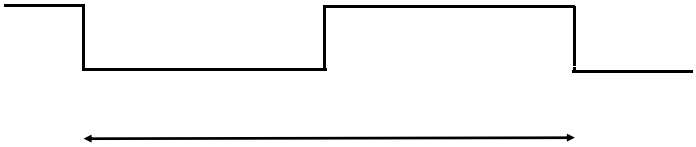
1) Memorizzare l'indirizzo dell'istruzione nel PC.
2) Leggere l'istruzione dalla memoria.
3) Aggiornare l'indirizzo in modo che in PC sia contenuto l'indirizzo dell'istruzione successiva.





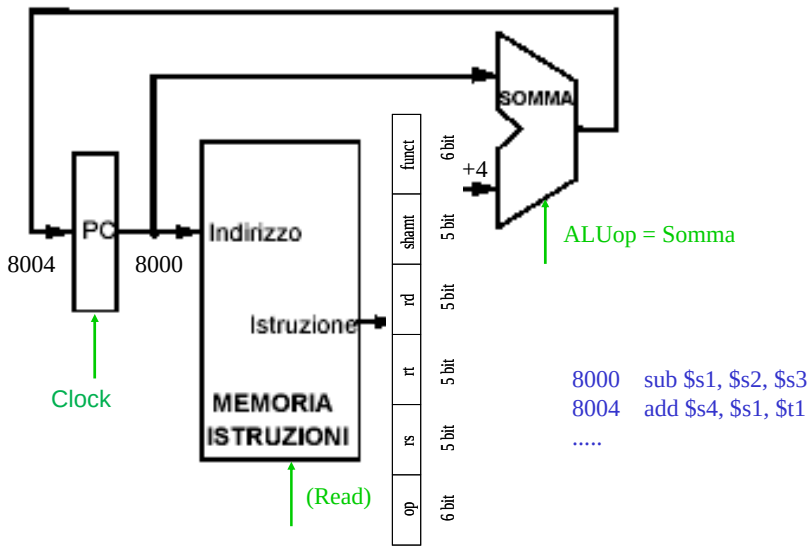

# Temporizzazione

1 istruzione per ciclo di clock. Temporizzazione del PC.

T > Tempo necessario per eseguire il cammino critico

# Circuito della fase di fetch






# Istruzioni di tipo R

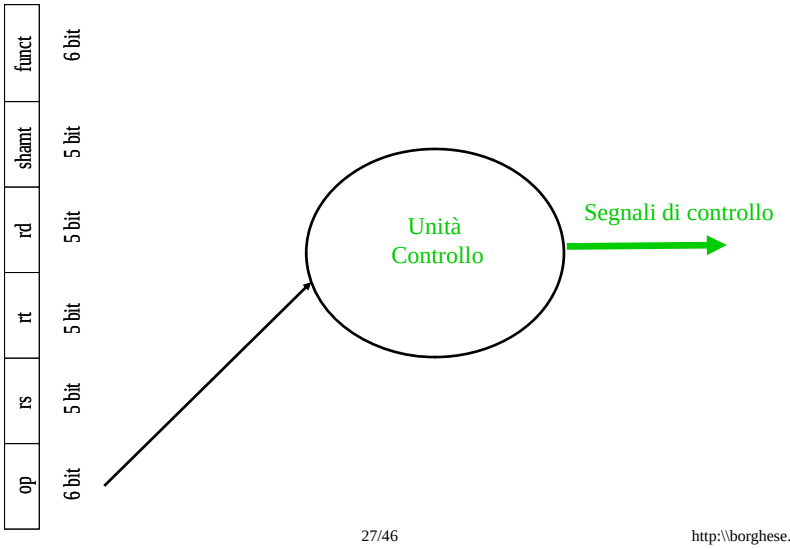
R

| op = 0 | rs = 18 | rt = 19 | rd = 17 | Shamt=0 | funct=34 |
|---|---|---|---|---|---|
| 6 bit | 5 bit | 5 bit | 5 bit | 5 bit | 6 bit |

sub $s1, $s2, $s3

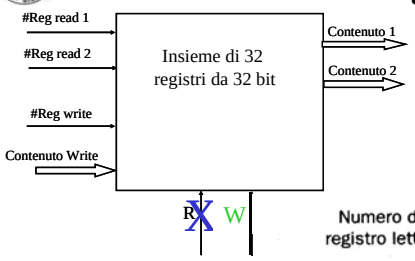
# Fase di decodifica

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



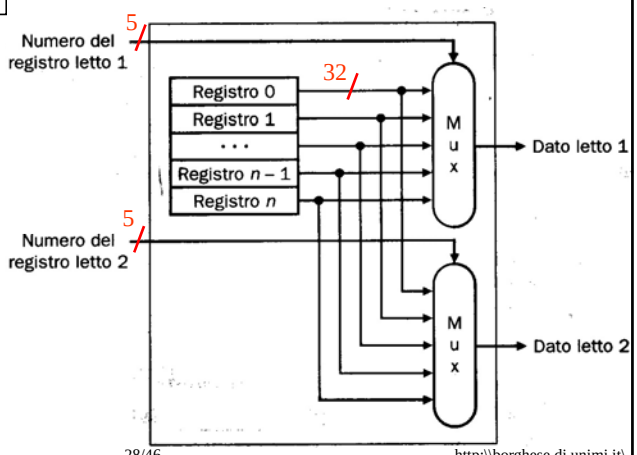



# Register file



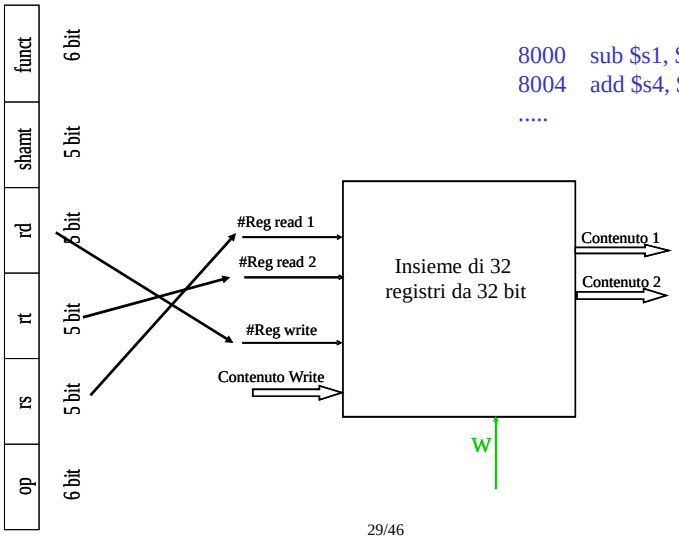

Banco di registri utilizzabile come memoria
Può essere scritto o letto.

Un mux per ogni porta di lettura.



# Lettura dei registri (istruzioni di tipo R)

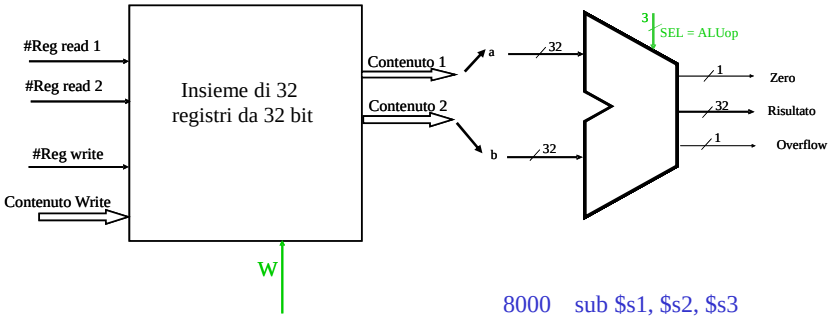
1) Leggo l'istruzione e genero i segnali di controllo opportuni.
2) Leggo il contenuto dei registri.




8000 sub $s1, $s2, $s3
8004 add $s4, $s1, $t1
.....

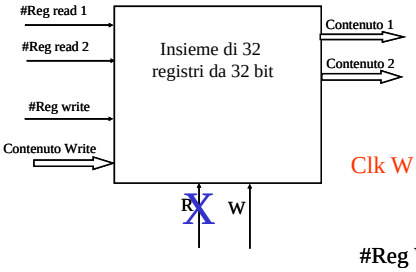


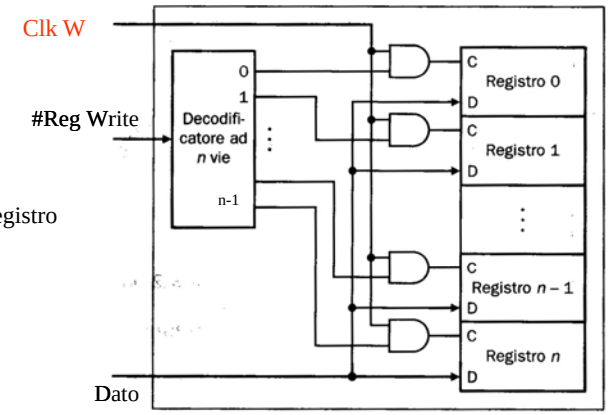
# Fase di Calcolo (tipo R)




8000 sub $s1, $s2, $s3
8004 add $s4, $s1, $t1
.....

# Porta di scrittura del register file




Ingresso C del latch dei registri:
Decodificatore per indirizzare il registro
*AND*
Comando Write

Ingresso D del latch dei registri:
Bit dato corrispondente.

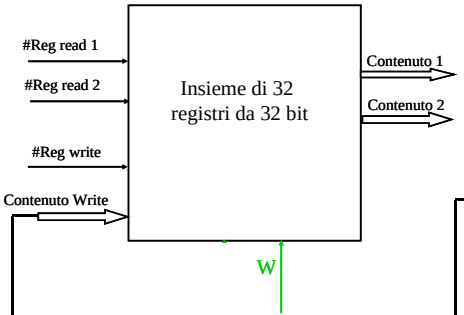
NB Utilizzo registri flip-flop in modo da potere leggere / scrivere nello stesso ciclo di clock (scrivo nel master nella fase di WB e leggo dallo slave in fase di decodifica. La commutazione da master a slave è pilotata dal clock.

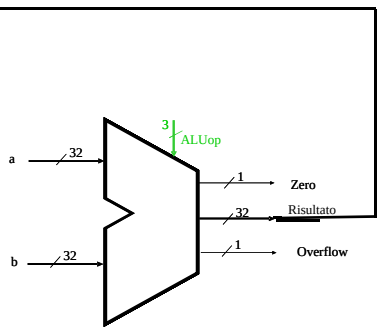
A

# Fase di Write back (tipo R)

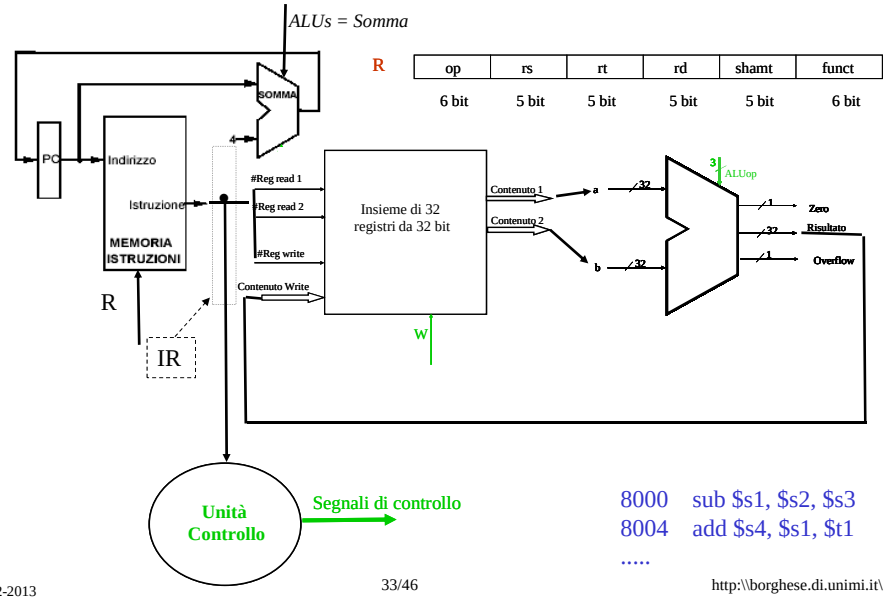
8000 sub $s1, $s2, $s3
8004 add $s4, $s1, $t1
.....

Risultato



# CPU per l'esecuzione completa di un'istruzione di tipo R




8000 sub $s1, $s2, $s3
8004 add $s4, $s1, $t1
.....



# Sommario

La CPU

Costruzione di una CPU per le istruzioni di tipo R

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

# Istruzioni di tipo I: lw/sw

I

| 100011 | 10001 | 10010 | 0000 0000 0001 0100 |
|---|---|---|---|
| 6 bit | 5 bit | 5 bit | 16 bit |

lw $s2, 20($s1)

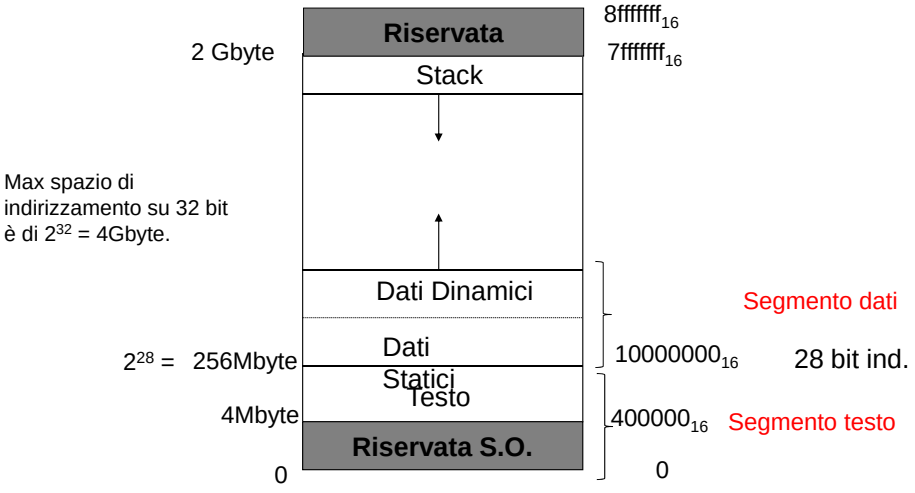
L'indirizzo di memoria sarà:

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



# Organizzazione logica della memoria

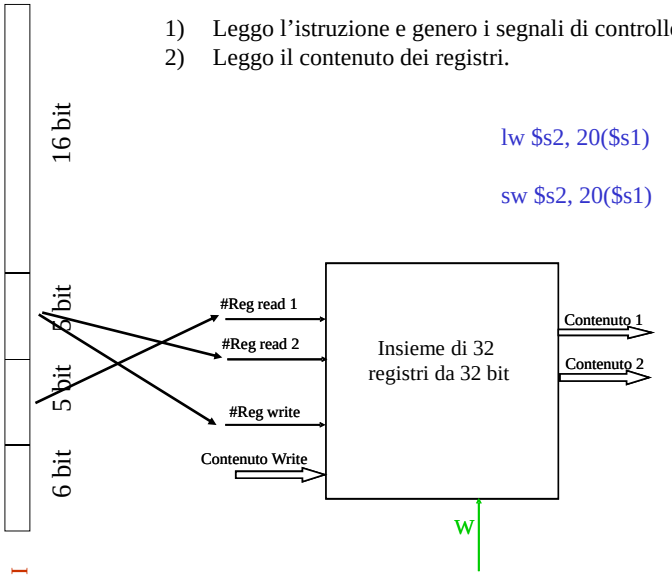


# Lettura dei registri (istruzioni di tipo I)

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

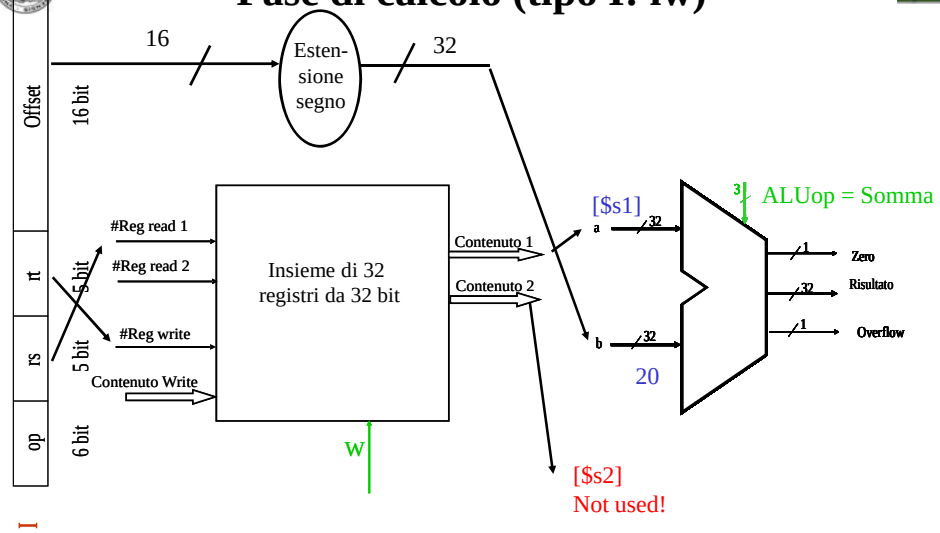
lw $s2, 20($s1)

sw $s2, 20($s1)





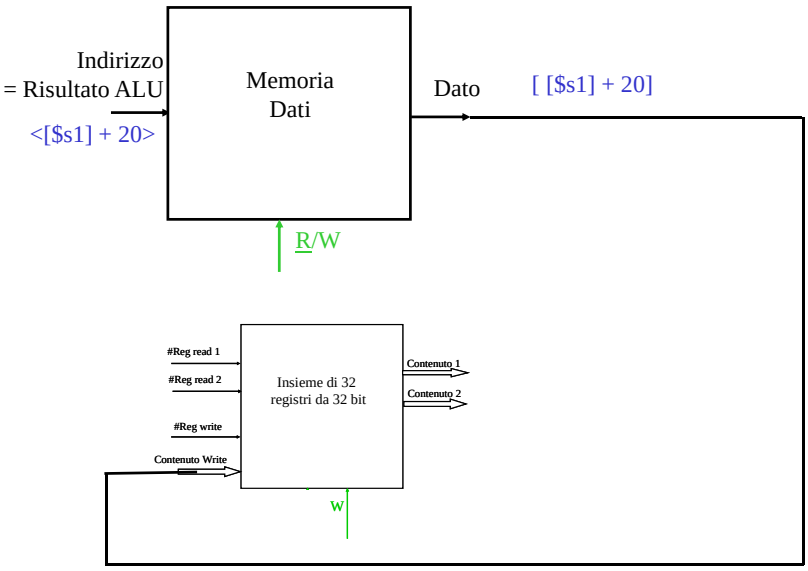

# Fase di calcolo (tipo I: lw)




Il Risultato è un indirizzo della memoria

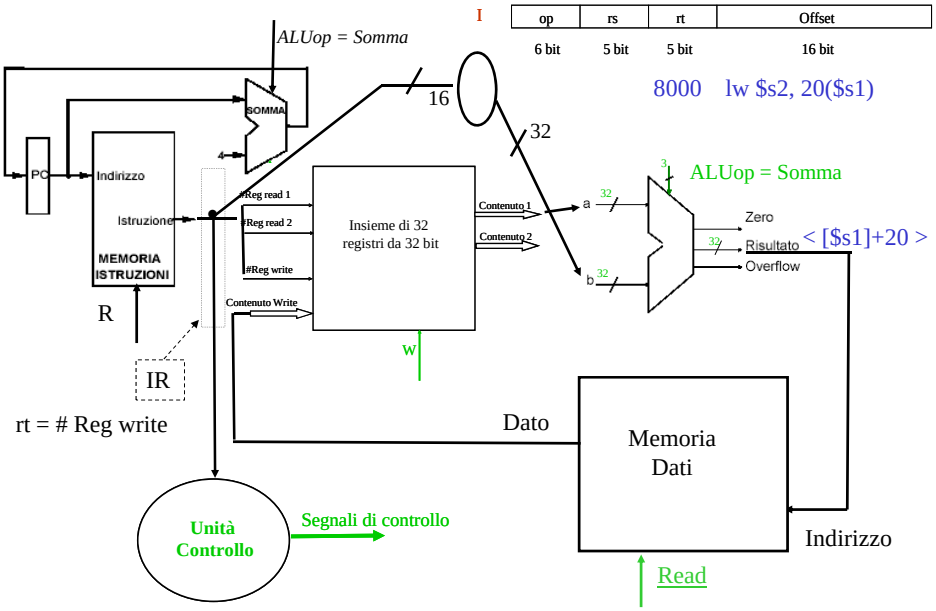
8000 lw $s2, 20($s1)

# Lettura della memoria + write-back

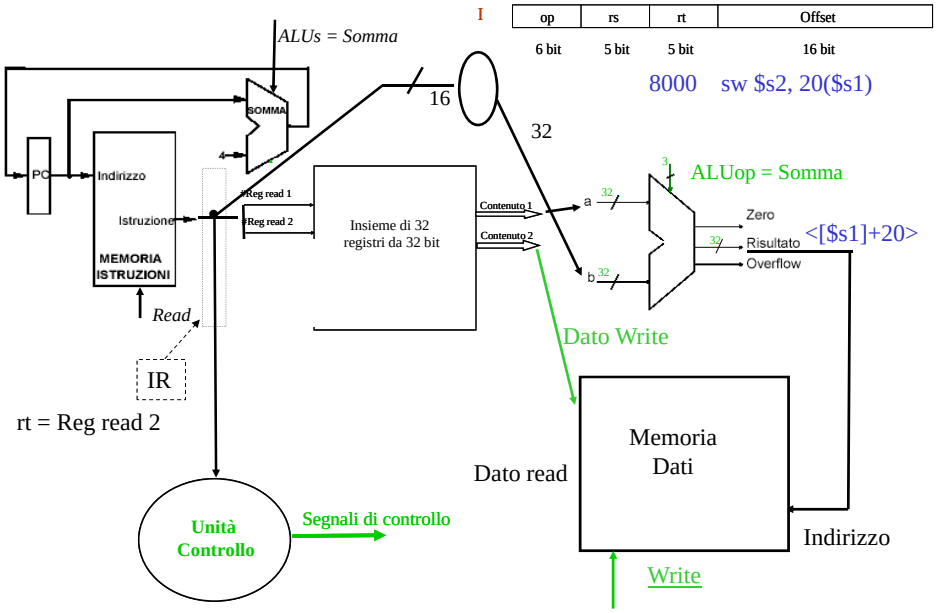





# CPU per l'esecuzione di una lw

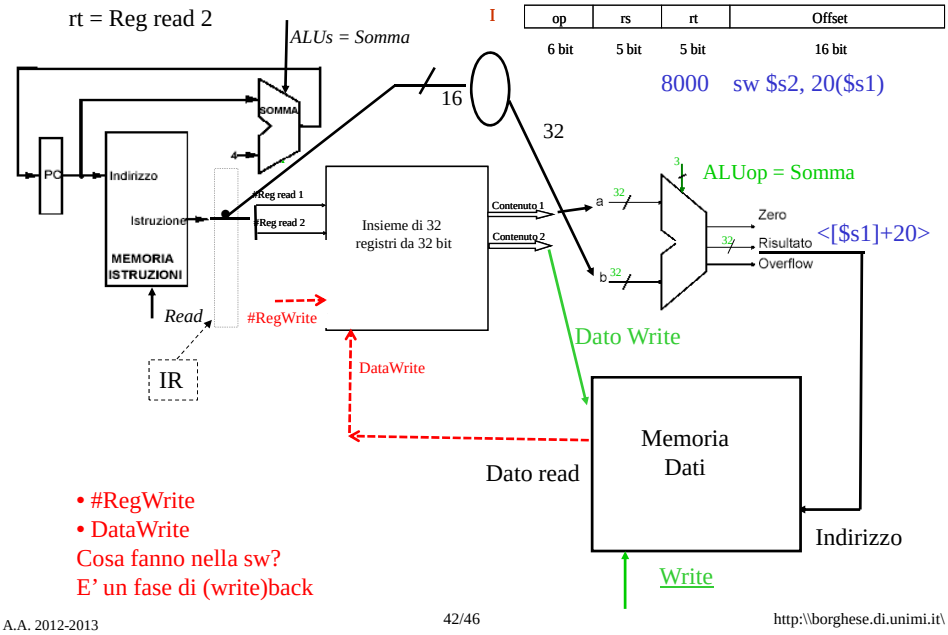


# CPU per l'esecuzione di una sw






# CPU per l'esecuzione di una sw

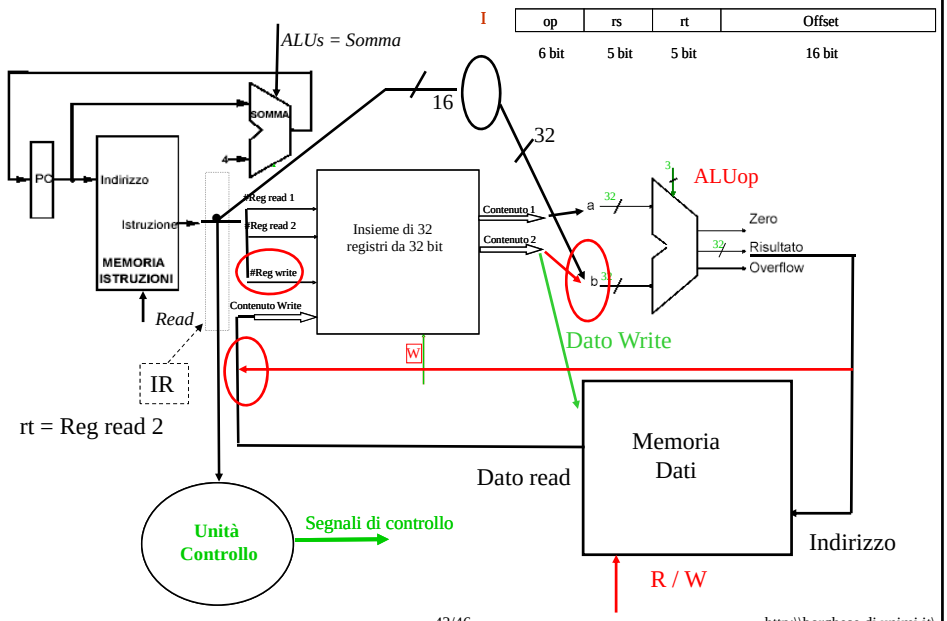



- #RegWrite
- DataWrite

Cosa fanno nella sw?

E' un fase di (write)back

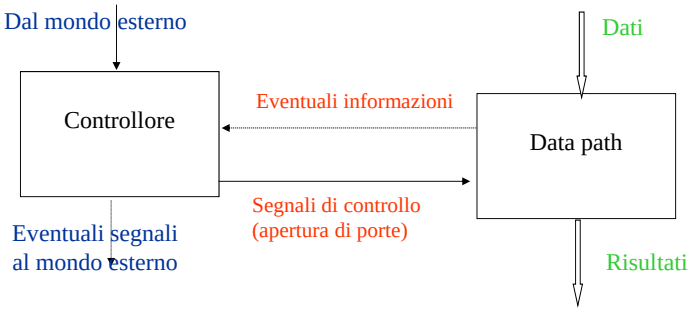
# CPU per l'esecuzione di una R, lw, sw






# Rapporto UC - Dati

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

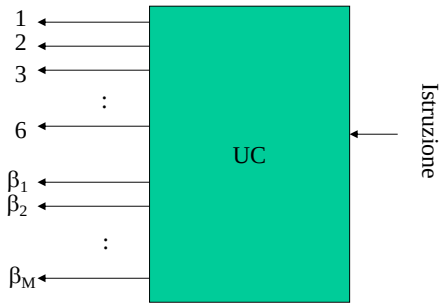



*Fase comune nel ciclo di esecuzione:*
- Fase di fetch
- Decodifica (generazione dei segnali di controllo)

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

# 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.






# Sommario

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).