Lezione 10

Architetture di elaborazione

- Architettura di Von Neumann
- Architetture moderne

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Che cos'è un elaboratore?

È una macchina che **automaticamente** esegue
una **sequenza di istruzioni** di **elaborazione di informazioni**

Architettura (struttura hardware) di un elaboratore: sono necessari:

- Un modulo in grado di elaborare informazioni (ALU)
- Un posto dove memorizzare le istruzioni da eseguire
- Un posto dove memorizzare i le informazioni da elaborare

→ MEMORIA

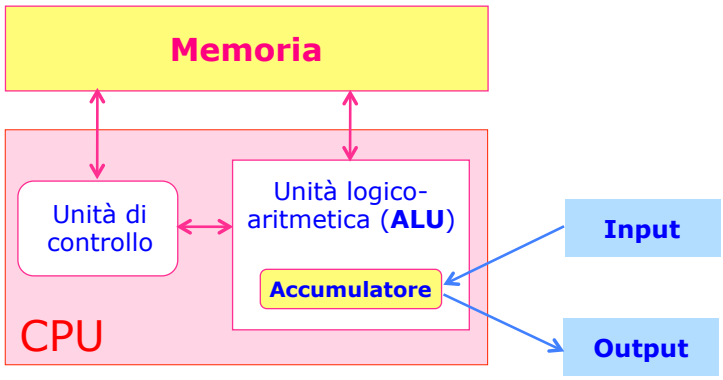
- Un meccanismo per trasferire informazioni da/verso l'esterno (IN/OUT)

Architettura di Von Neumann

- **Architettura di VON NEUMANN**:
  - **Dati** e **Istruzioni** sono contenute in una **memoria** leggibile e scrivibile
    Accesso al contenuto della memoria avviene in base alla posizione (indirizzo)
  - Esecuzione **sequenziale:** da un'istruzione alla seguente
  - I risultati di un'operazione sono sempre contenuti nell'unico registro della CPU chiamato **accumulatore**

Architettura di Von Neumann:
*(architettura **ad accumulatore**)*

***ALU e UC** contenute nella CPU*
***un solo registro**: accumulatore*



Architetture di tipo CISC

Architetture moderne:

- **CISC**: **C**omplex **I**nstruction **S**et **C**omputer
  Elevata complessità delle istruzioni eseguibili
  Elevato numero di istruzioni disponibili

- Numerose **modalità di indirizzamento** per gli operandi dell'ALU
  - possono provenire da registri oppure da memoria
  - Indirizzamento diretto, indiretto, con registro base, ecc.

- Ricchezza del set di istruzioni ➔ istruzioni con tanti formati differenti, 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
  - Rallentano i tempi di esecuzione delle operazioni.
  - Elevata profondità dell'albero delle porte logiche, utilizzato per la decodifica.

CISC vs. RISC

Statistica di utilizzo delle istruzioni:

La maggior parte del tempo di elaborazione viene impiegato per eseguire poche istruzioni semplici.

IDEA: → Ottimizzo le istruzioni semplici, trascuro il resto
→ Set di istruzioni ridotto, solo istruzioni semplici **RISC**

Rank	Instruction	Average Execution Frequency (%)
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%

Architetture RISC

Architetture moderne:

- **RISC**: **R**educed **I**nstruction **S**et **C**omputer

IDEA: eseguire soltanto istruzioni semplici

- Operazioni complesse scomposte in serie di istruzioni semplici da eseguire in un ciclo-base ridotto, ma ottimizzato.
- *CPU* semplice → si riducono i tempi di esecuzione delle singole istruzioni
- Risultato: **poche istruzioni, ma molto veloci!**
  → raggiungo / supero approccio CISC

- Esempi RISC:
  - Architettura semplice: ad esempio, Load-Store – gli operandi dell'*ALU* possono provenire solo dai registri, *non* dalla memoria
  - Dimensione **fissa** delle istruzioni → semplice gestione della fase di prelievo ***(fetch)*** e della decodifica delle istruzioni

Tipi di architetture

Esempi di architetture moderne di CPU:

- **Accumulator** (1 register = 1 indirizzo di memoria)

1 address	**add A**	`acc ← acc + mem[A]`
1+x address	**add x A**	`acc ← acc + mem[A + x]`

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

0 address	**add**	`Mem[ToS] ← Mem[ToS] + Mem[ToS−1]`

- **Register-addressed Memory** (tanti indirizzi di memoria quanti sono i registri: indirizzamento indiretto)

2 address	**add A B**	`Mem(A) ← Mem(A) + Mem(B)`
3 address	**add A B C**	`Mem(A) ← Mem(B) + Mem(C)`

- **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*

- I registri ad uso generale in una CPU sono insufficienti a memorizzare tutte le variabili di un programma

  Quando un registro deve essere utilizzato per contenere un'altra informazione (es. un'altra variabile), si può trasferire l'informazione precedente in memoria.

*Architetture **LOAD/STORE:***

Simili all'architettura di Von Neumann, solo con più registri (***Register File***)

- Gli **operandi dell'ALU possono provenire soltanto dai registri** contenuti nella CPU e **non** possono provenire direttamente **dalla memoria**.
- Sono necessarie apposite istruzioni di:
  - *caricamento (**LOAD**)* dei dati da memoria ai registri;
  - *memorizzazione (**STORE**)* dei dati dai registri alla memoria.

Architettura load/store – elementi principali

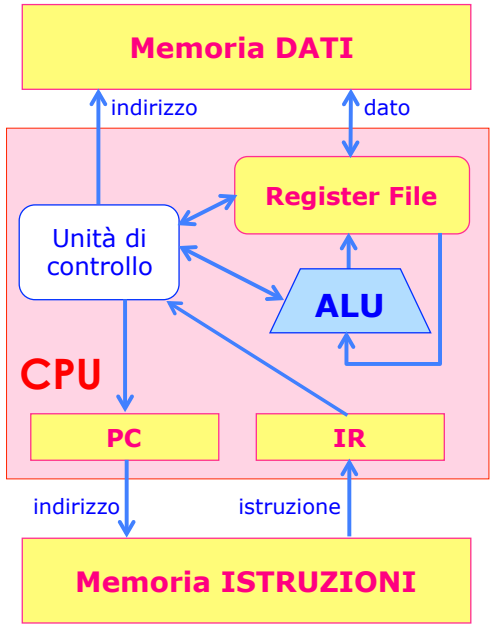
Architettura "load/store" – la CPU contiene:

- **Registri:**
  - Banco di registri di uso generale **(Register File)**

    Memoria ad accesso rapido, in cui memorizzare i dati di utilizzo più frequente
  - **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
  - **Instruction Register (IR)**

    Contiene l'istruzione in corso di esecuzione.
- **Arithmetic Logic Unit – (ALU)**
  - Effettua le operazioni aritmetico-logiche fra registri
- **Unità di controllo**
  - "Cervello" della CPU
  - Coordina i flussi di informazione, in funzione dell'istruzione in corso
  - Seleziona l'operazione opportuna della ALU.



Sommario

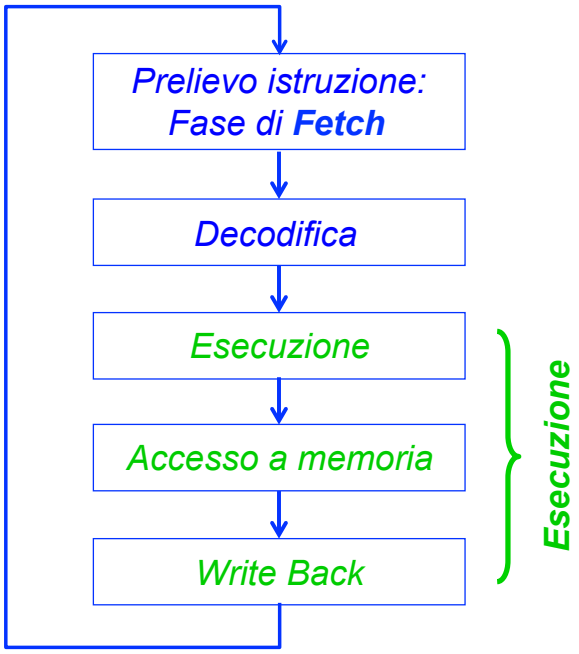
- Architettura di Von Neumann e architetture moderne.
- La CPU – il ciclo di esecuzione di un'istruzione.

Unità centrale di elaborazione (CPU)

- *Compito della CPU: eseguire in sequenza le istruzioni che costituiscono il programma assegnato all'elaboratore.*
  - *ciclo di esecuzione di istruzioni*

- Eseguire un'istruzione significa:
  - **acquisire** l'istruzione da eseguire: **FETCH**
  - **interpretare** l'istruzione e i dati a disposizione: **DECODIFICA**
  - eseguire calcoli / scelte / trasferimenti informazione **ESECUZIONE**

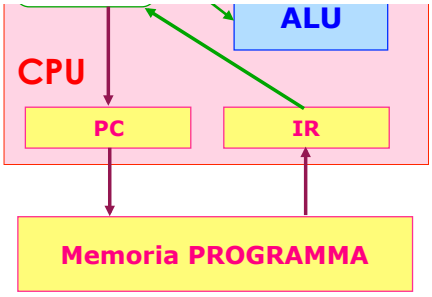
Ciclo esecuzione istruzione

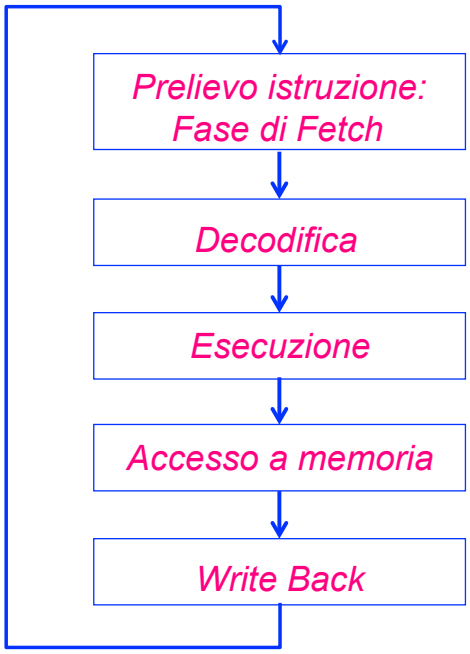


- **Istruzioni e dati** risiedono nella **memoria principale**
- L'esecuzione di un programma inizia quando il registro **PC** punta alla prima istruzione del programma
- **MIPS**: suddivisione dell'esecuzione in tre sottofasi

Lettura istruzione (Fetch)

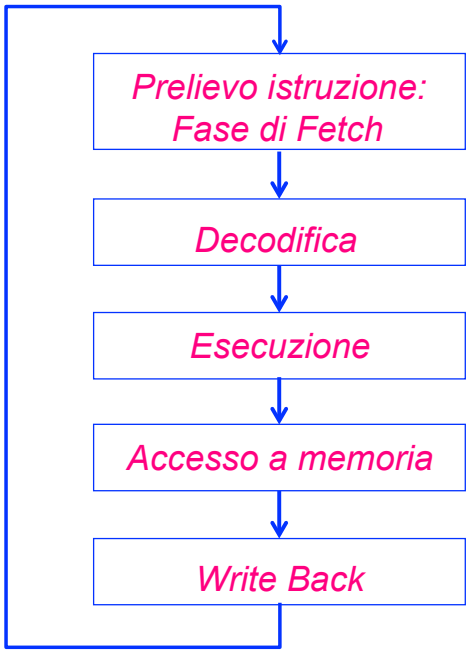
- Fase di **FETCH**
  - Il segnale di controllo per la **lettura** viene inviato alla memoria
  - Trascorso il tempo necessario all'accesso in memoria, la parola indirizzata (istruzione) viene letta dalla memoria e trasferita nel registro **IR**





Decodifica dell'istruzione

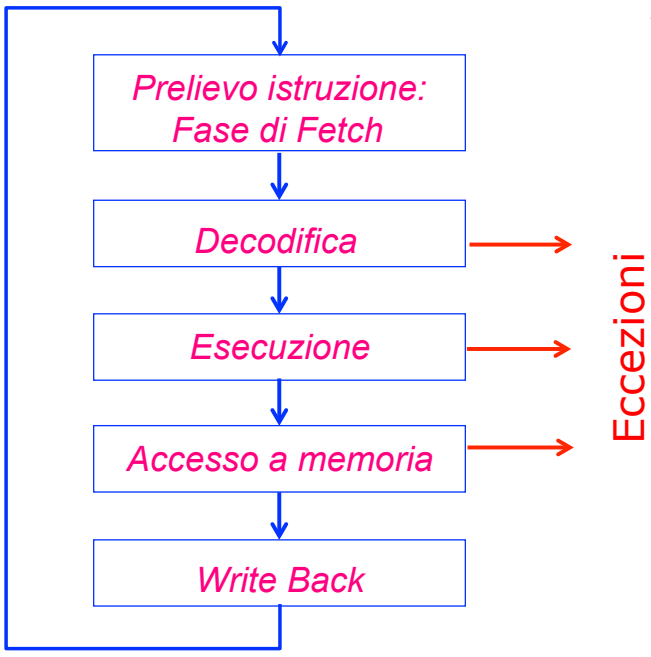
- DECODIFICA: l'istruzione contenuta nel registro IR viene decodificata per essere eseguita
- Predisposizione della CPU all'esecuzione dell'istruzione:
  - Se l'istruzione è aritmetico-logica, è necessario recuperare gli operandi
- Apertura delle vie di comunicazione appropriate
  - Lettura/scrittura di registri
  - Lettura/scrittura verso memoria
  - Lettura/scrittura dall'esterno (I/O)
  - …



Write-back: *Riscrittura risultato*

- Esecuzione:
  - Vengono selezionati i circuiti combinatori appropriati per l'esecuzione dell' istruzione e determinate in fase di decodifica.
- Accesso a MEMORIA:
  - SCRITTURA: se il risultato dell'operazione deve essere posto in memoria, esso viene inviato al bus dati. L'indirizzo viene posto sul bus indirizzi e si comanda una scrittura (WRITE) in memoria.
  - LETTURA: se il dato va prelevato dalla memoria, l'indirizzo viene posto sul bus indirizzi e si attiva una lettura (READ), che porta la parola sul bus dati.
- Write-back:
  - Il risultato dell'operazione può essere memorizzato in un registro.
- Chiusura ciclo:
  - Mentre viene eseguita un'istruzione, il contenuto del PC viene incrementato in modo da puntare alla prossima istruzione da eseguire.

Ciclo di esecuzione di un'istruzione



- Il normale flusso di esecuzione di un programma può essere interrotto da *Eccezioni*:

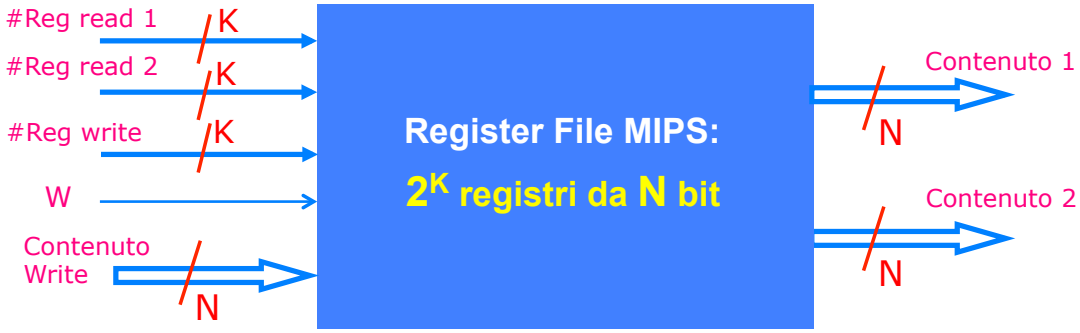
ESTERNE: un segnale di interruzione esterno *(INTERRUPT)* per una richiesta di intervento.

INTERNE: Un segnale di errore di calcolo (es. divisione per 0)

Registri della CPU: Register File

- **Register file**: elemento di una CPU che contiene tutti i registri di uso generale **(general purpose registers)**
  - Banco di registri: 2^K **registri** da **N bit** ciascuno
  - Solitamente, **2 porte di uscita, 1 in ingresso**

- Operazioni:
  - SELEZIONE: fornendo in ingresso il numero del registro (#reg)
  - LETTURA: non modifica il contenuto del registro selezionato
  - SCRITTURA: Inserisce <ContenutoWrite> nel registro selezionato (comando: W)



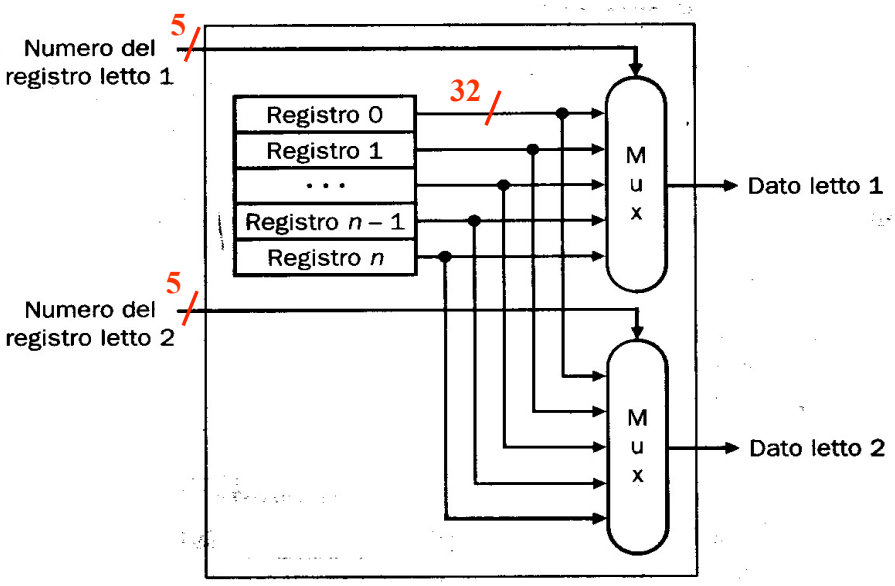
Register file (CPU **MIPS**)

- **Register File**: famiglia di CPU **"MIPS"**:
  - **32 registri da 32 bit**
  - **1** linea di **ingresso**
  - **2** linee di **uscita**

Register File MIPS: Porta di **LETTURA**

Doppia porta di **lettura**

- 2 MUX di selezione registro
  - 2 registri possono essere letti contemporaneamente



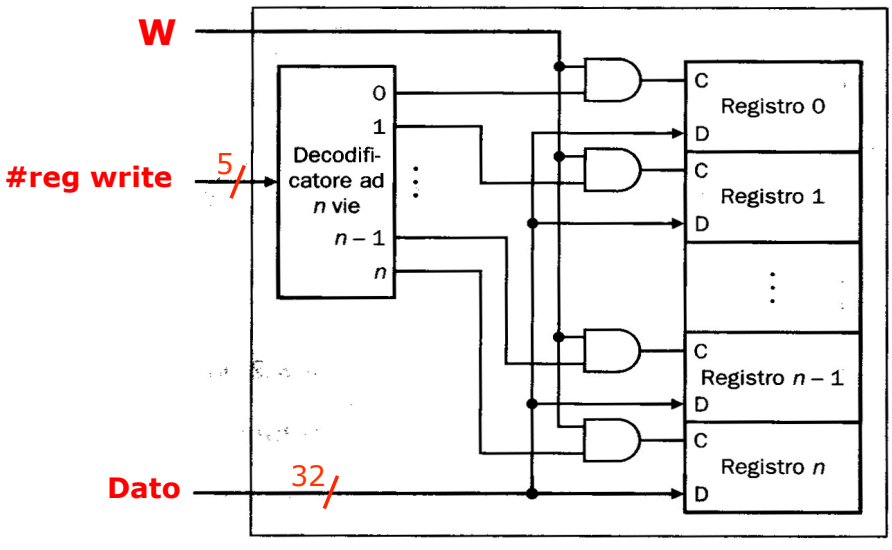
Register file MIPS: Porta di **SCRITTURA**

Porta di scrittura

Sugli ingressi di ogni registro fornisco:

- **Ingresso C:** **(decoder(i) AND W)**
  - Se no ad ogni impulso su W scriverei in tutti i registri

- **Ingresso D:**
  - Dato da scrivere nel registro (32 bit)



Indirizzi nella memoria principale

MEMORIA

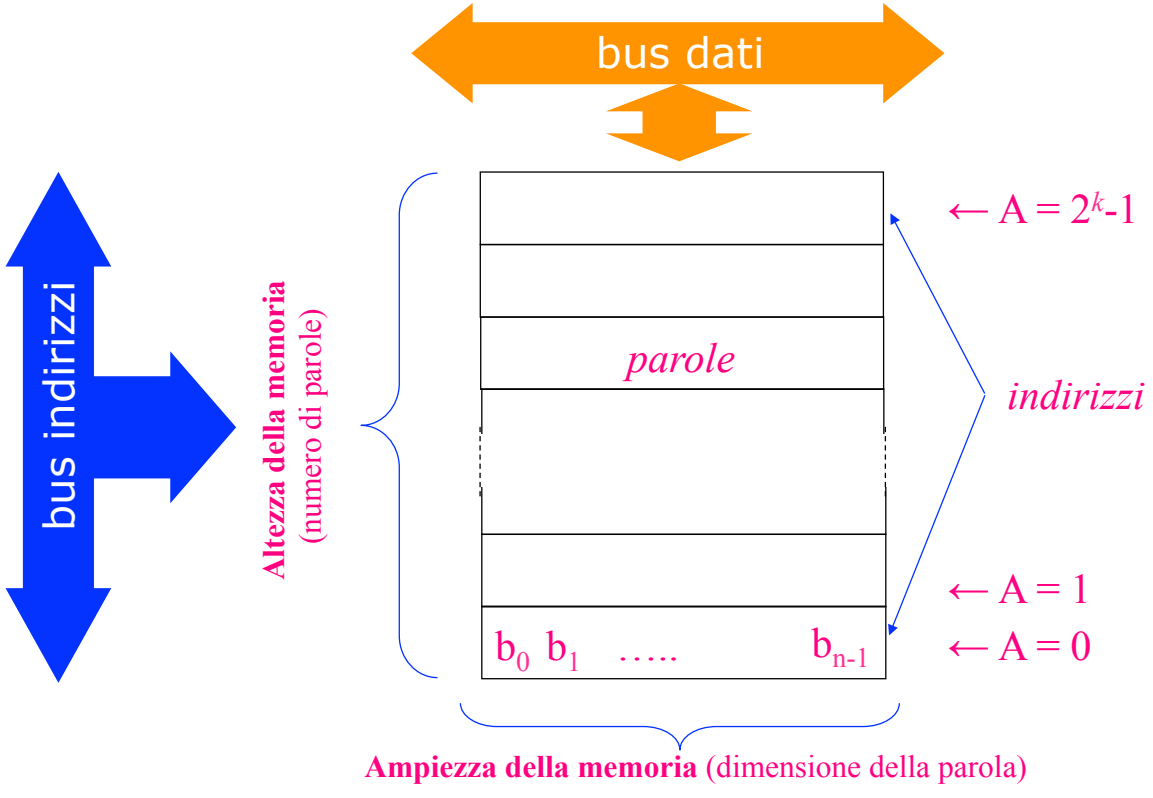
- La memoria è organizzata in ***2^K parole (word)*** di ***N bit***
  - ***N: ampiezza*** *della memoria*
  - ***2^K: altezza*** *della memoria*

- In genere, la dimensione della parola di memoria coincide con la dimensione dei registri della *CPU (CPU word)*
  - Le operazioni di *load/store* avvengono in un **singolo ciclo**

- **Capacità della memoria =** *(N.words)* x *(dim.word [bytes])*
  - ***k = 32 → (2^{32}= 4 Gwords)*** x ***(32 bit = 4 bytes) = 16 Gbytes***

- *Random Access Memory –* ***RAM***
  - **Il tempo di accesso alla memoria** è **fisso e indipendente dalla posizione della parola** alla quale si vuole accedere.

Indirizzi nella memoria principale



La memoria nel MIPS

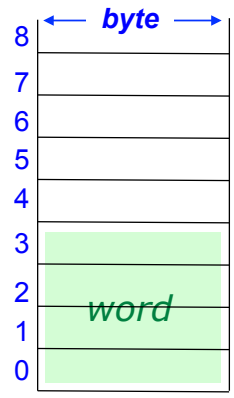
MEMORIA

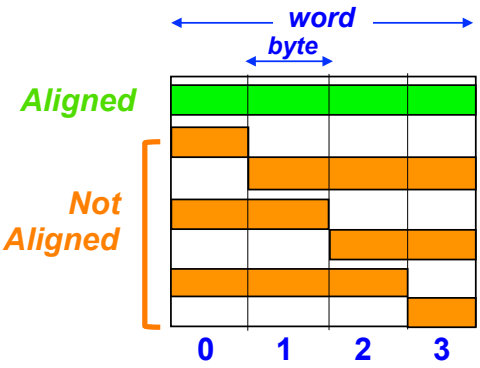
La memoria è vista come un grande

array uni-dimensionale **di bytes**
array uni-dimensionale **di words**

Indirizzo di memoria **univoco** per ciascun **byte**

- Indirizzamento al byte
  - l'indice punta ad un byte di memoria
- Indirizzamento alla parola
  - La memoria è vista come un array di PAROLE
  - 32 bit: gli indirizzi di parole consecutive differiscono di un fattore 4





Alignment (allineamento)

- L'indirizzamento al byte mi permette di scrivere a cavallo tra 2 parole
- Le parole devono avere indirizzi MULTIPLI della loro dimensione

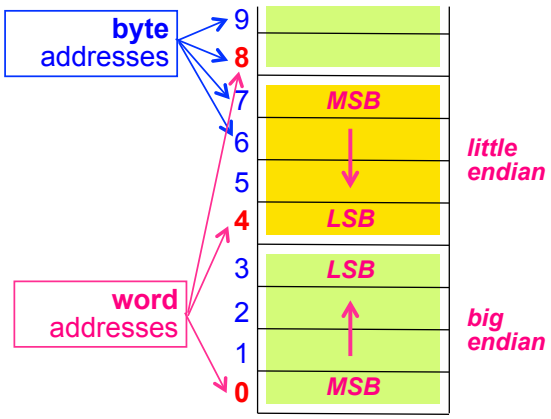
"Endianess" e "alignment"

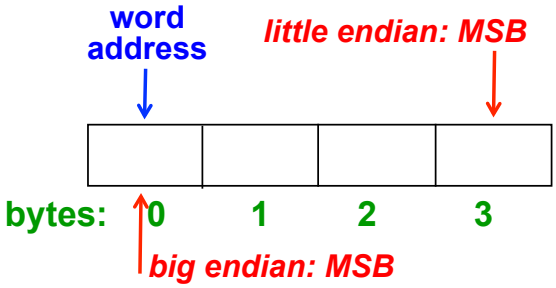
MEMORIA

Endianess: come si rappresenta un dato su **più bytes**

- **Big Endian:**
  Word address = address of MSB (most significant byte)
  - Motorola 68k, PowerPC, MIPS, Sparc, HP-PA

- **Little Endian:**
  Word address = address of LSB (least significant byte)
  - Intel x86, DEC Vax, DEC Alpha





Disposizione Big-Endian

Big-endian

- I byte sono numerati partendo dalla posizione **più** significativa;
- Sono big-endian: MIPS, PowerPC, ARM

Indirizzo di parola ➔ indirizzo del suo **byte più significativo**

	Indirizzo di byte			
Parola: **0**	0	1	2	3
Parola: **4**	4	5	6	7
Parola: **8**	8	9	10	11
…				
…				
Parola: $2^k - 4$	2^k-4	2^k-3	2^k-2	2^k-1

IEEE 754

Disposizione Little-Endian

Little-endian:

- I byte sono numerati partendo dalla posizione **meno** significativa;
- Sono little-endian: famiglia INTEL

Indirizzo di parola ➔ indirizzo del suo **byte meno significativo**

	Indirizzo di byte			
Parola: **0**	3	2	1	0
Parola: **4**	7	6	5	4
Parola: **8**	11	10	9	8
…				
…				
Parola: $2^k - 4$	2^k-1	2^k-2	2^k-3	2^k-4

IEEE 754

MIPS32 memory: 32-bit words – big-endian

Indirizzo word	**Indirizzo byte**
0	**0**
1	**4**
2	**8**
$2^k/4$	**2^k**

0	1	2	3
4	5	6	7
8	9	10	11
2^K	$2^K + 1$	$2^K + 2$	$2^K + 3$