

# Architettura degli Elaboratori

*Università degli Studi di Padova*



*Facoltà di Scienze MM.FF.NN.*

**Corso di Laurea in Informatica**

**Seconda parte**

*docente: Alessandro Sperduti*

## Informazioni Generali

- Lucidi ed esercizi disponibili in formato elettronico  
<http://www.math.unipd.it/~sperduti/architettura1.html>
- Modalità d'esame : 
  - di norma: scritto *obbligatorio* con orale *opzionale*
  - in casi eccezionali (**es. studente che copia durante la prova d'esame scritta**): scritto ed orale *obbligatori*
- Registrazione dell'esame: 
  - **SOLO** nelle date deputate alla registrazione (una data per appello d'esame)
  - entro l'anno accademico (30 Settembre 2010), altrimenti si **PERDE** il voto

## Obiettivi della seconda parte del Corso

- Descrizione dell'architettura degli elaboratori *in ambito locale* dal punto di vista funzionale e tecnologico
- Introdurre *i principi e le tecniche generali* (per aumentare l'efficienza e facilità d'uso degli elaboratori), validi indipendentemente dai dettagli implementativi di ogni specifica architettura
- Testi di riferimento:



- W. Stallings. **Architettura e organizzazione dei calcolatori 6/E - Progetto e prestazioni** Pearson Education Italia
- J L Hennessy & D A Patterson  
**Computer Architecture A Quantitative Approach**  
ISBN 1-55860-329-8 (consultazione)

## Contenuto del Corso



1. **Struttura e funzione della CPU** [Ch. 12]
2. **Processori RISC** [Ch. 13]
3. **Memoria cache** [Ch. 4]
4. **Memoria interna** [Ch. 5]
5. **Memoria esterna** [Ch. 6]
6. **Input/Output** [Ch. 7]

## Struttura e funzione della CPU

Cercheremo di rispondere alle seguenti domande:

- Come si differenziano le CPU dal punto di vista dei registri e del loro uso ?
- Come può una CPU organizzare i calcoli relativi alla esecuzione delle istruzioni in modo da essere più efficiente ?

**efficiente:**

utilizzare al meglio tutte le componenti della CPU in modo da non avere componenti inattive (inefficienza)

**pipelining:**



- ogni istruzione, quando è eseguita, “passa” attraverso stadi separati di esecuzione (ad es.: fetch, decodifica, calcolo indirizzi,...);
- *idea:* mantenere tutti gli stadi attivi eseguendo più istruzioni, come in una catena di montaggio (ogni stadio lavora contemporaneamente su una istruzione diversa)

## Processori RISC



Cercheremo di rispondere alle seguenti domande:

- Quali sono le istruzioni più frequentemente utilizzate dai programmi scritti nei linguaggi ad alto livello ?
- Si può utilizzare la risposta alla prima domanda per decidere quale deve essere il set delle istruzioni da implementare per una CPU ?

**Reduced Instruction Set Computer (RISC):**

- un ridotto insieme di istruzioni, tutte dello stesso formato
- un elevato numero di registri o l'impiego di un compilatore che ne ottimizza l'uso
- particolare attenzione all'ottimizzazione della pipeline

# Memoria Cache



Cercheremo di rispondere alla seguente domanda:

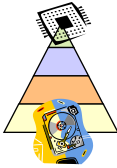
- Memorie RAM:

- quelle veloci sono molto costose e di capacità limitata;
- quelle di grossa capacità sono lente ma economiche;

come è possibile combinare l'uso di questi due tipi di memorie in modo da avere una memoria RAM:

- di grossa capacità
- abbastanza veloce
- economica ?

Gerarchia di memoria:



- la CPU richiede i dati necessari alla memoria veloce (Cache)
- se la Cache non li contiene, li richiede alla memoria più lenta che contiene tutti i dati
- quindi dati riferiti frequentemente si troveranno in Cache

# Memoria Interna



Cercheremo di rispondere alle seguenti domande:

- Che tipo di memoria, e quale tecnologia, viene usata per la memoria Cache ?

**SRAM (static RAM)**

- Che tipo di memoria, e quale tecnologia, viene usata per la RAM più lenta ma più capiente ?

**DRAM (dynamic RAM)**

- Come si possono organizzare più moduli di DRAM per avere un accesso più veloce ed efficiente ?

**DRAM sincrona, DRAM RamBus**



## Input/Output



Cercheremo di rispondere alle seguenti domande:

- Come vengono organizzate e gestite le operazioni di input/output fra la CPU ed i dispositivi esterni (disco rigido, tastiera, video, ...)?
- Cosa è un modulo di I/O? Quali funzionalità svolge? Quale è la sua struttura?
- Quali sono le alternative possibili di gestione dell' I/O?
  - I/O da programma
  - I/O driven (guidato da interrupt)
  - accesso diretto alla memoria (DMA)
- Cosa è un canale o processore di I/O?

## Memoria Esterna








Cercheremo di rispondere alle seguenti domande:

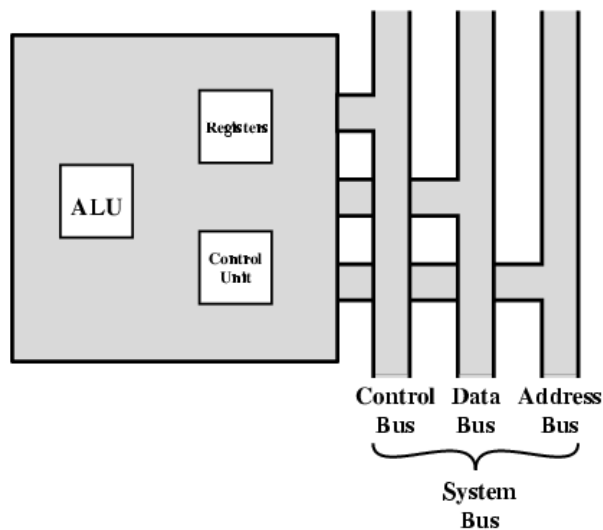
- Come viene memorizzata l'informazione in un disco rigido?
- Come si accede all'informazione in un disco rigido?
- Come si possono usare più dischi rigidi in parallelo in modo da recuperare più velocemente le informazioni e preservarla da guasti?
  - Redundant Array of Independent Disk (RAID)**
- Come viene memorizzata l'informazione in un CD-ROM o DVD?
- Con quale tecnologia?

# Struttura CPU

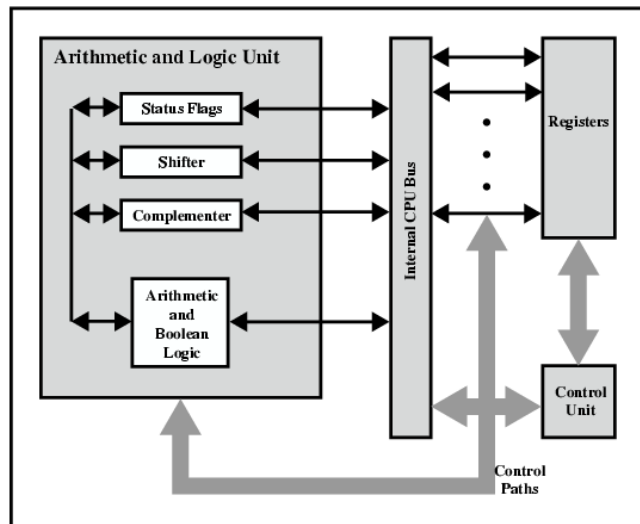
Compiti CPU:

- Prelevare istruzioni 
- Interpretare istruzioni 
- Prelevare dati 
- Elaborare dati 
- Scrivere (memorizzare) dati 

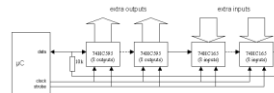
# CPU con bus di sistema



## Struttura interna CPU



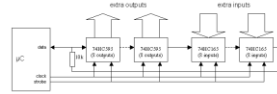
## Registri



- CPU ha bisogno di uno “spazio di lavoro” dove memorizzare i dati
- Questo “spazio di lavoro” è costituito dai **registri**
- Numero e funzioni svolte dai registri varia a seconda dell’impianto progettuale della CPU
- Scelta progettuale molto importante
- I registri costituiscono il livello più alto della così detta “Gerarchia della memoria”



## Registri



- Registri utente
  - usati dal **“programmatore”** per memorizzare internamente alla CPU i dati da elaborare
- Registri di controllo e di stato
  - usati dall’unità di controllo per monitorare le operazioni della CPU
  - usati dai programmi del Sistema Operativo (SO) per controllare l’esecuzione dei programmi

## “programmatore”

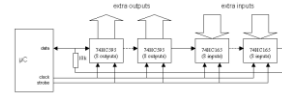


1. Umano che programma in assembler
2. Compilatore che genera codice assembler a partire da un programma scritto in un linguaggio ad alto livello (C, C++, Java,...)

**Ricordarsi che un programma in assembler è trasformato in codice macchina dall’assemblatore (+ linker) che trasforma il codice mnemonico delle istruzioni in codice macchina**



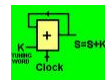
## Registri visibili all'utente: registri utente



- Ad uso generale
- Per la memorizzazione di dati
- Per la memorizzazione di indirizzi
- Per la memorizzazione di codici di condizione

## Registri ad uso generale

- Possono essere veramente ad uso generale
- ...oppure dedicati a funzioni particolari
- Possono essere usati per contenere dati o indirizzi
- Dati
  - Ad esempio: accumulatore
- Indirizzi
  - Ad esempio: indirizzo base di un segmento di memoria



## Segmento di memoria



- La memoria principale può essere organizzata, dal punto di vista logico (cioè concettuale), come un insieme di segmenti o spazi di indirizzamento multipli:
  - “visibili” al “programmatore”, che riferisce logicamente una locazione di memoria riferendo il segmento e la posizione della locazione all’interno del segmento:  
es. segmento 4, locazione 1024
  - come supporto a questa “visione” della memoria, occorre poter indicare **dove**, all’interno della memoria fisica, inizia il segmento (*base*) e la sua **lunghezza** (*limite*)  
es. il segmento 4 ha base = 00EF9445<sub>hex</sub> e limite = 4MB
  - quindi occorrono dei registri dove memorizzare tali informazioni

## Registri ad uso generale

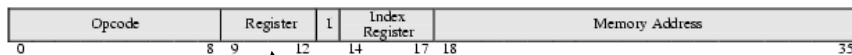
- Registri veramente ad uso generale
  - Aumentano la flessibilità e le opzioni disponibili al programmatore “a basso livello”
  - Aumentano la dimensione dell’istruzione e la sua complessità (perché ?)
- Registri specializzati
  - Istruzioni più piccole e più veloci
  - Meno flessibili



## Perché aumenta dimensione e ↑ complessità ?

- Facciamo l'esempio di istruzioni a formato fisso:

PDP-10



I = indirect bit

Formato più  
lungo! 37 bit

4 bit  $\rightarrow 2^4 = 16$  registri generali  
Se si avessero 32 registri generali  
sarebbero necessari 5 bit di indirizzamento

## Quanti registri generali?

- Tipicamente tra 8 e 32
- Meno di 8 = più riferimenti (accessi) alla memoria principale (perché ?)
- Più di 32 non riducono i riferimenti alla memoria ed occupano molto spazio nella CPU
- Nelle architetture RISC tipicamente si hanno più di 32 registri generali



## Perché più accessi ?



ESEMPIO: supponiamo di dover calcolare:

```
mem[4] = mem[0]+mem[1]+mem[2]+mem[3]
```

```
mem[5] = mem[0]*mem[1]*mem[2]*mem[3]
```

```
mem[6] = mem[5]-mem[4]
```

<b>4 registri</b>	<b>7 op</b>	<b>6 registri</b>	<b>7 op</b>
R0 ← mem[0];	<b>5 mem</b>	R0 ← mem[0];	<b>4 mem</b>
R1 ← mem[1];		R1 ← mem[1];	
R2 ← mem[2];		R2 ← mem[2];	
R3 ← mem[3];		R3 ← mem[3];	
R0 ← R0+R1;		R4 ← R0+R1;	
R0 ← R0+R2;		R4 ← R2+R4;	
R0 ← R0+R3;		R4 ← R3+R4;	
R1 ← R1*R2;		R5 ← R0*R1;	
R1 ← R1*R3;		R5 ← R2*R5;	
R2 ← mem[0];		R5 ← R3*R5;	
R1 ← R1*R2;		R0 ← R5-R4;	
R0 ← R1-R0;			

## Quanto lunghi (in bit) ?

- Abbastanza grandi da contenere un indirizzo della memoria principale
- Abbastanza grandi da contenere una “full word”
- E’ spesso possibile combinare due registri dati in modo da ottenerne uno di dimensione doppia
  - Es.: programmazione in C
    - double int a;
    - long int a;

## Registri per la memorizzazione di Codici di Condizione

- Insiemi di bit individuali
  - es. Il risultato dell'ultima operazione era zero
- Possono essere letti (implicitamente) da programma
  - es. “Jump if zero” (salta se zero)
- Non possono (tipicamente) essere impostati da programma

## Registri di Controllo e di Stato

- Program Counter (PC)
- Instruction Register (IR)
- Memory Address Register (MAR)
- Memory Buffer Register (MBR)



## Program Status Word

- Un insieme di bit
- Include Codici di Condizione
  - Segno dell'ultimo risultato
  - Zero
  - Riporto
  - Uguale
  - Overflow
  - Abilitazione/disabilitazione Interrupt
  - Supervisore



## Modo Supervisore

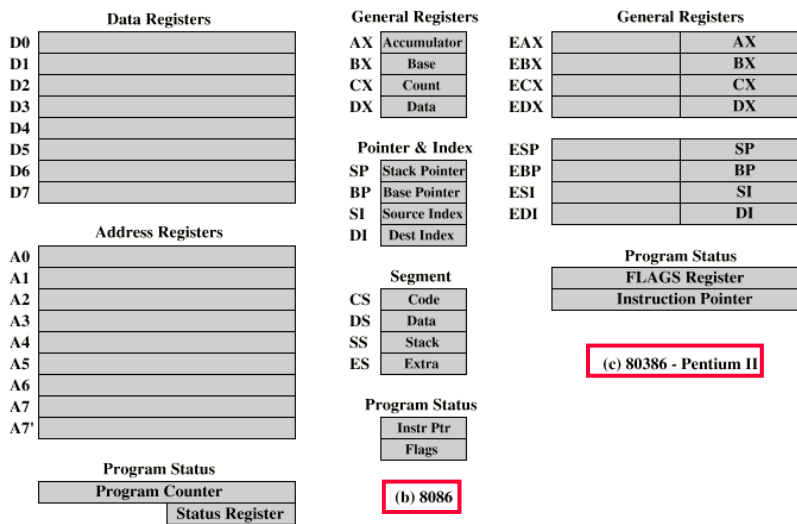


- Permette al sistema operativo di utilizzare le procedure del Kernel, che agiscono su componenti critiche del sistema
- In particolare permette l'esecuzione di istruzioni "privilegiate"
- Disponibile esclusivamente al sistema operativo
- Non disponibile all'utente programmatore
- Lo studierete in dettaglio nel corso di Sistemi Operativi

## Altri registri

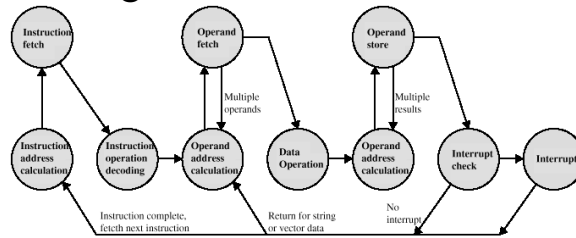
- Ci possono essere registri che puntano a:
  - Process control blocks (sistemi operativi)
  - Interrupt Vectors (sistemi operativi)
  - Tabella delle pagine della memoria virtuale
- La progettazione della CPU e quella dei sistemi operativi sono strettamente correlate

## Esempi di Organizzazione di Registri



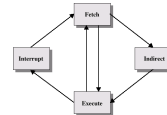
## Ciclo esecutivo delle istruzioni: Fetch/Execute

- Lo avete già visto



- Stallings, Capitolo 3
- Ne vediamo una versione revisionata

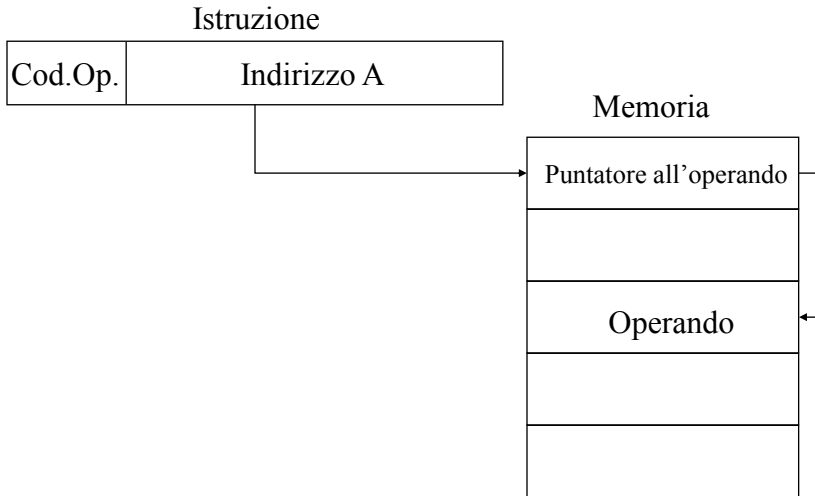
## Indirettezza



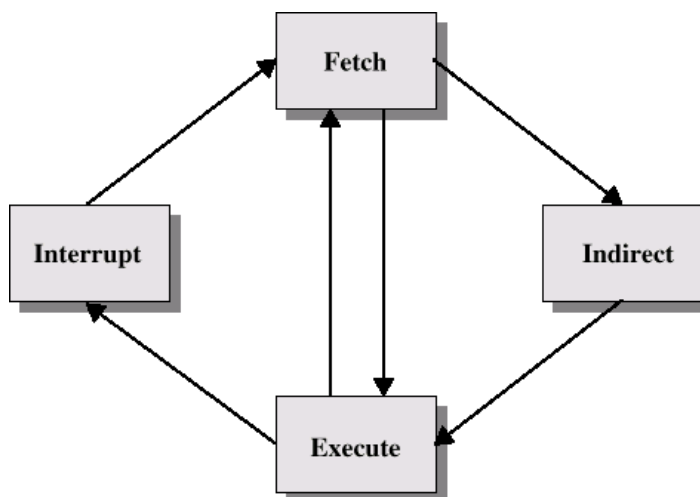
- Per recuperare gli operandi di una istruzione può essere necessario accedere alla memoria
- La modalità di **indirizzamento indiretto** per specificare la locazione in memoria degli operandi richiede più accessi in memoria
- L'indirettezza si può considerare come un sottociclo del ciclo fetch/execute

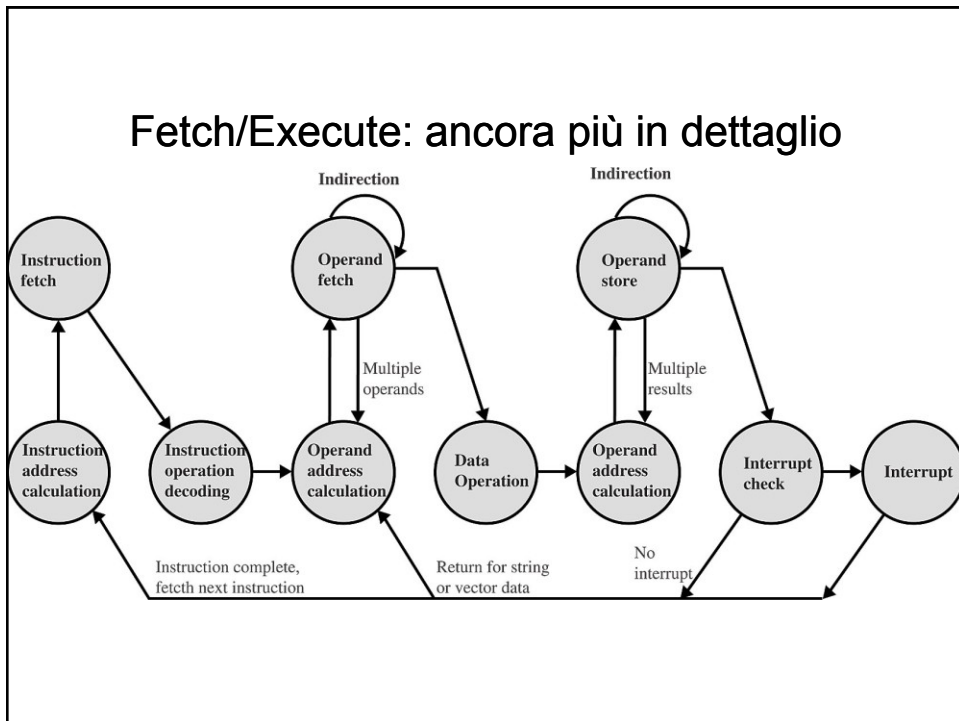


# Indirizzamento indiretto



## Introduzione della Indirettezza (indirect)





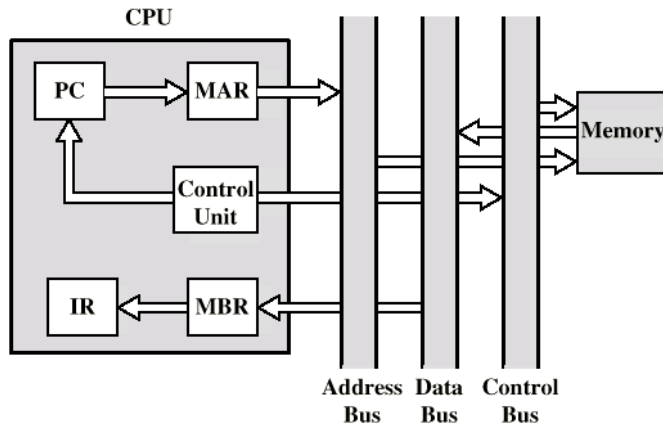
## Flusso dei dati (Instruction Fetch)

Dipende dalla architettura della CPU, in generale:



- Fetch
  - PC contiene l'indirizzo della istruzione successiva
  - Tale indirizzo viene spostato in MAR
  - L'indirizzo viene emesso sul bus degli indirizzi
  - La unità di controllo richiede una lettura in memoria principale
  - Il risultato della lettura in memoria principale viene inviato nel bus dati, copiato in MBR, ed infine in IR
  - Contemporaneamente il PC viene incrementato

## Flusso dei dati (Diagramma di Fetch)



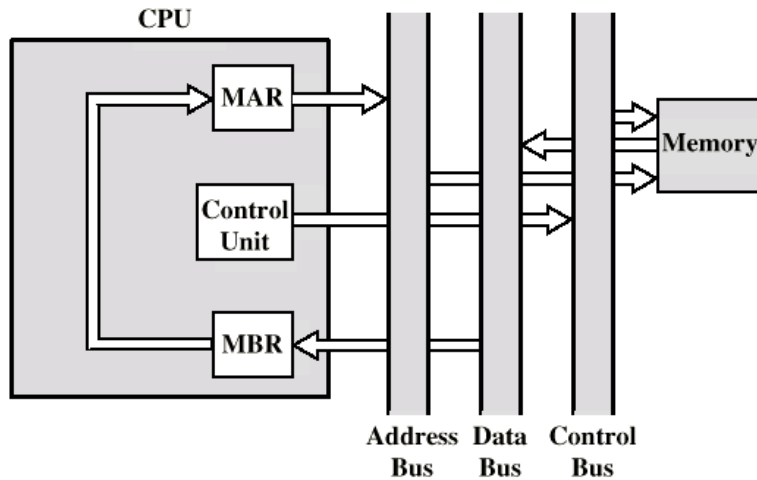
MBR = Memory buffer register  
 MAR = Memory address register  
 IR = Instruction register  
 PC = Program counter

## Flusso dei dati (Data Fetch)



- IR è esaminato
- Se il codice operativo della istruzione richiede un indirizzamento indiretto, si esegue il ciclo di indirettezza
  - gli N bit più a destra di MBR vengono trasferiti nel MAR
  - L'unità di controllo richiede la lettura dalla memoria principale
  - Il risultato della lettura (indirizzo dell'operando) viene trasferito in MBR

## Flusso dei dati (Diagramma di Indirettezza)



## Flusso dei dati (Execute)



- Può assumere molte forme
- Dipende dalla istruzione da eseguire
- Può includere
  - lettura/scrittura della memoria
  - Input/Output
  - Trasferimento di dati fra registri e/o in registri
  - Operazioni della ALU

## Flusso dei dati (Interrupt)



Semplice e prevedibile:

- Contenuto corrente del PC deve essere salvato per permettere il ripristino della esecuzione dopo la gestione dell'interruzione
  - Contenuto PC copiato in MBR
  - Indirizzo di locazione di memoria speciale (es. stack pointer) caricato in MAR
  - Contenuto di MBR scritto in memoria
- PC caricato con l'indirizzo della prima istruzione della routine di gestione della interruzione
- Fetch della istruzione puntata da PC

## Flusso dei dati (Diagram. di Interrupt)

