

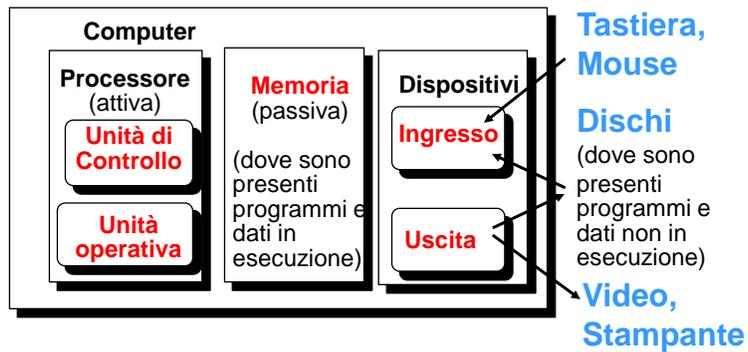
## Architettura dei Calcolatori Elettronici

### Caratteristiche di un calcolatore elettronico

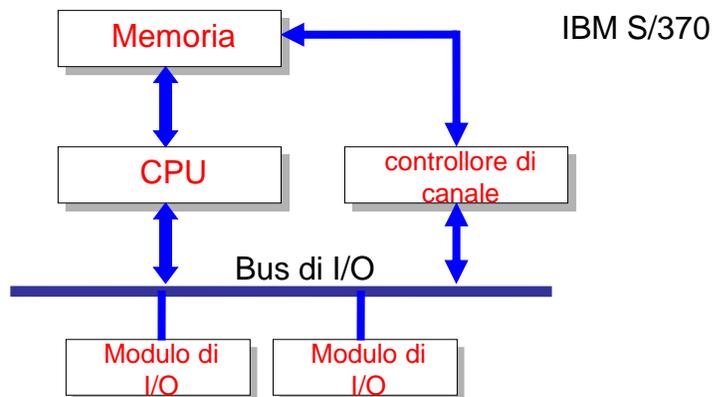


- Capacità di eseguire sequenze di istruzioni memorizzate
- **Calcolatore** = Unità di Elaborazione + Unità di Controllo
  - 1. Preleva le istruzioni dalla memoria
  - 2. Interpreta i codici di istruzione
  - 3. Effettua le azioni che questi prevedono
- **Programma** = Insieme organizzato di istruzioni

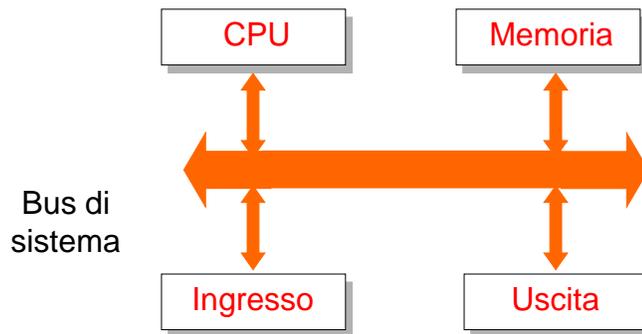
# Componenti di un Computer



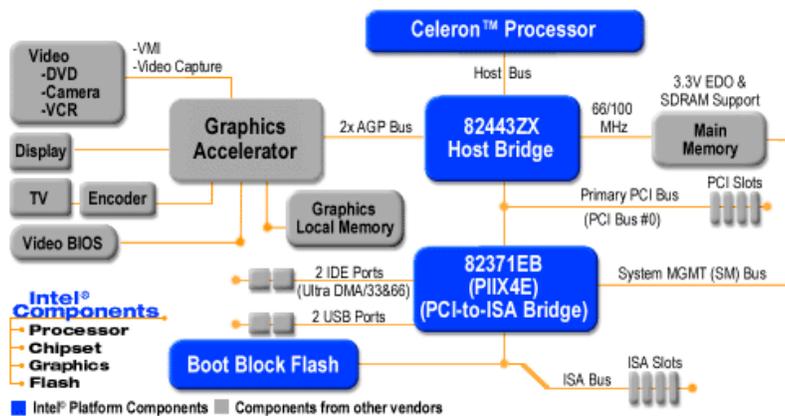
# Organizzazione di un calcolatore elettronico



## Organizzazione di un calcolatore elettronico

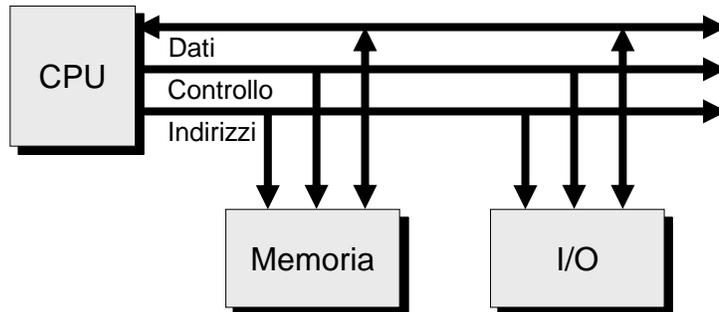


## Struttura PC corrente

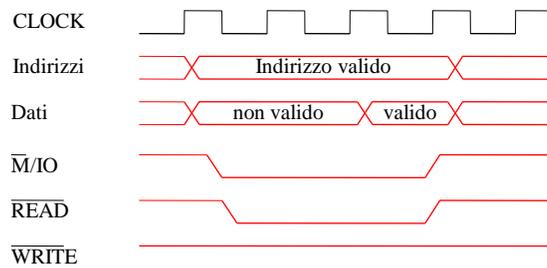


## Schema di riferimento

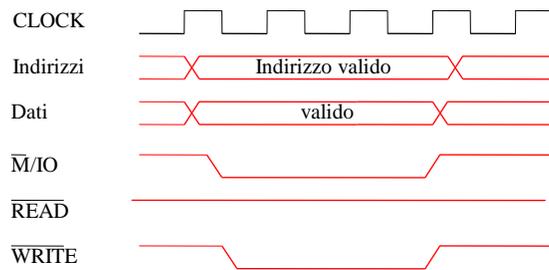
- Per il momento lo schema di riferimento sarà quello di figura
- Corrisponde allo schema dei PC anni 80
- Tuttora in largo uso nei sistemi di controllo



## Ciclo di lettura



## Ciclo di Scrittura



## Architettura di Von Neuman

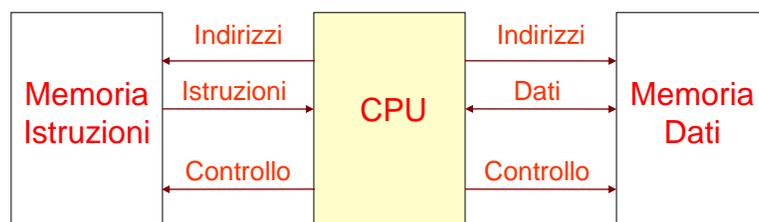


- Memoria indifferenziata per dati o istruzioni.
- Solo l'interpretazione da parte di CPU stabilisce se una data configurazione di bit è da riguardarsi come un dato o come un'istruzione

## Collo di Bottiglia Von Neumann

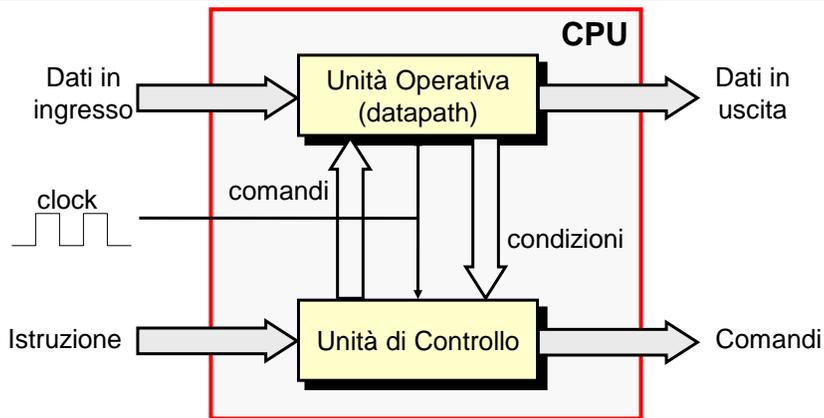
- L'organizzazione di Von Neumann è quella più popolare
- Consente al processore di manipolare i programmi in modo più semplice
- **Svantaggi**
  - La limitata larghezza di banda della memoria ha un impatto negativo sulla velocità di esecuzione dell'applicazione
  - Questo fenomeno è noto come “Von Neumann bottleneck”

## Architettura Harvard



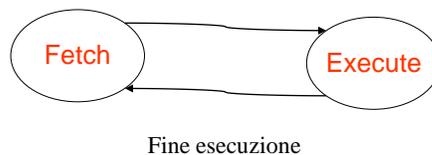
- Le istruzioni e i dati sono memorizzati in memorie distinte
- E' principalmente utilizzata nei processori ad alte prestazioni e nelle architetture dedicate per applicazioni di elaborazione digitale dei segnali (DSP)

# Struttura della CPU



## Fetch-Esecuzione

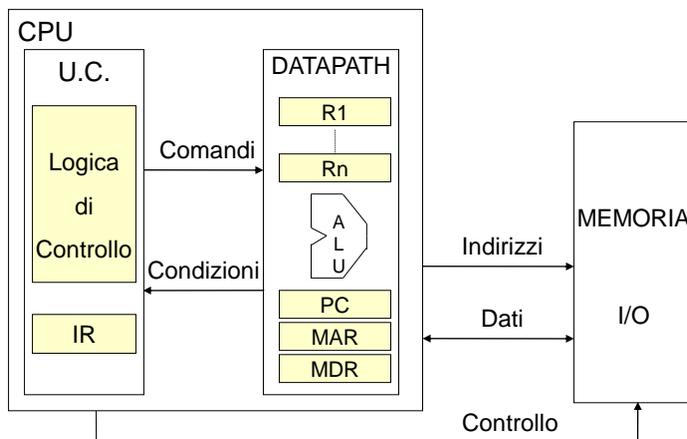
- **Fetch e decodifica:**
  - Prelievo e decodifica dell'istruzione
  - Fase comune a tutte le istruzioni
- **Esecuzione:**
  - Fase in cui vengono eseguite le azioni previste dal codice di operazione
  - Fase diversa da istruzione a istruzione



## Il Programma

- Programma = Sequenza di istruzioni
- Le istruzioni sono in memoria a indirizzi contigui
- Occorre un registro per memorizzare l'indirizzo della prossima istruzione da eseguire
  - Usualmente denominato **Program Counter (PC)**
- A termine dell'esecuzione di un'istruzione, PC deve puntare alla prossima istruzione
  - Le istruzioni sono a lunghezza fissa (stesso # di bytes)
    - PC è incrementato di una quantità pari a tale numero
  - Le istruzioni hanno lunghezza variabile
    - PC deve essere incrementato di volta in volta della dimensione in byte dell'istruzione appena eseguita
  - Le istruzioni di salto hanno l'effetto di aggiornare il PC con l'indirizzo di destinazione del salto

## Elementi fondamentali della CPU



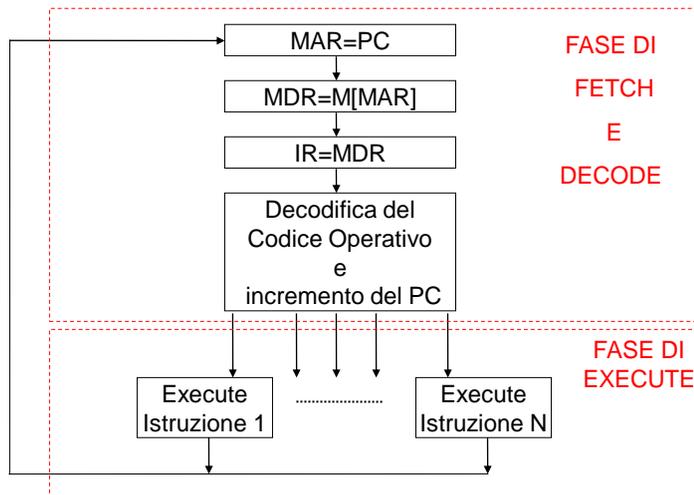
## Registri di CPU

- **IR:** Usato per contenere l'istruzione in corso di esecuzione
  - Caricato in fase di fetch
  - Rappresenta l'ingresso che determina le azioni svolte durante la fase di esecuzione
- **PC:** Tiene traccia dell'esecuzione del programma
  - Contiene l'indirizzo di memoria in cui è memorizzata la prossima istruzione da eseguire

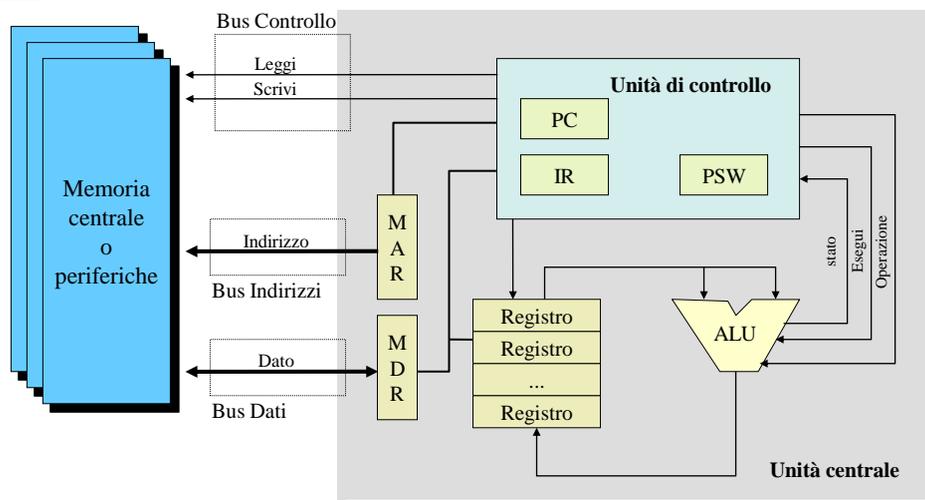
## Registri di CPU

- **MAR:** contiene l'indirizzo della locazione di memoria da leggere o scrivere
  - La dimensione di MAR determina l'ampiezza dello spazio di memoria fisica
  - Dalla fine degli anni '80 vengono prodotti microprocessori con bus indirizzi a 32 bit
- **MDR:** Registro attraverso il quale viene scambiata l'informazione tra la memoria e la CPU
  - Tradizionalmente la dimensione di MDR dà la misura del grado di parallelismo della macchina (8, 16, 32, 64 bit)
- **R0, R1, ..., Rn:** Registri di uso generale

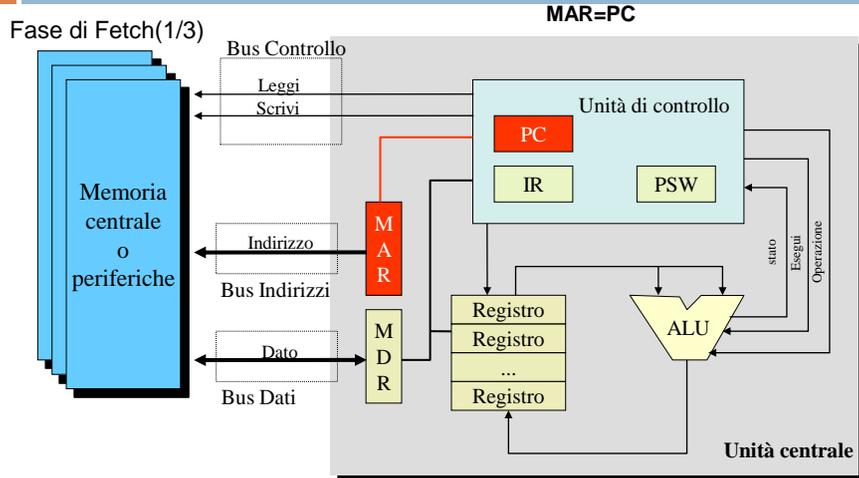
## Diagrammi a Stati della CPU durante l'esecuzione delle istruzioni



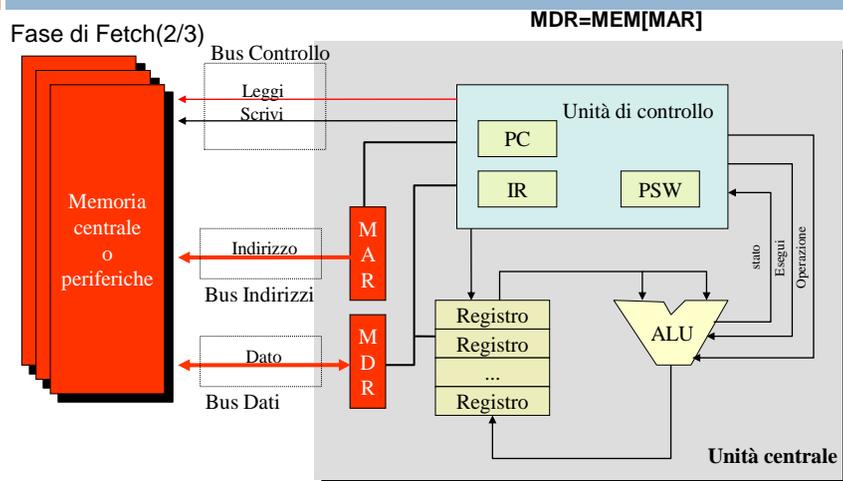
## Struttura Semplicata di una CPU



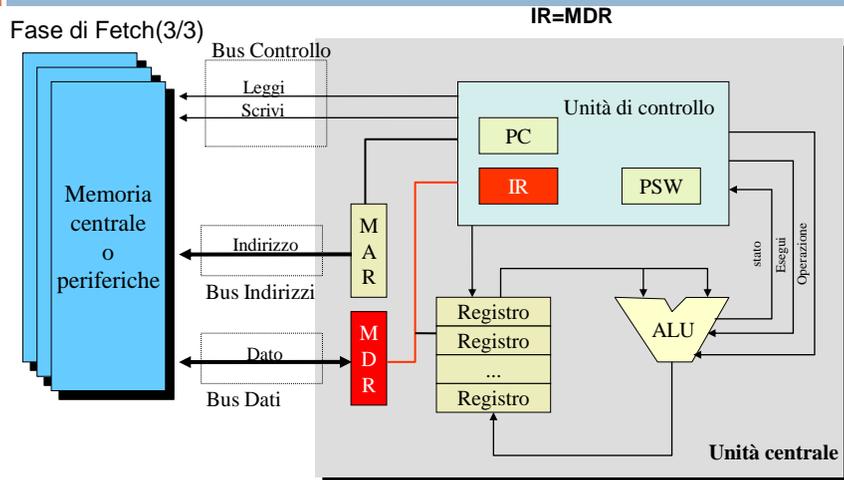
## Esempio: Lettura dalla Memoria



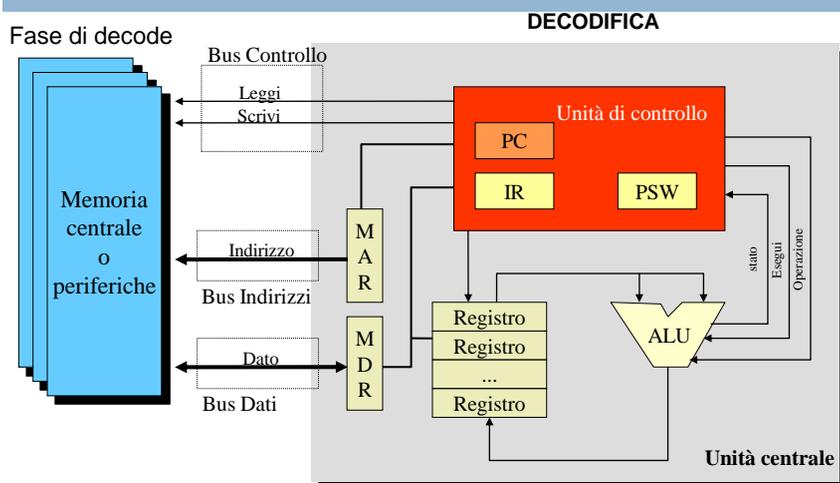
## Esempio: Lettura dalla Memoria



## Esempio: Lettura dalla Memoria

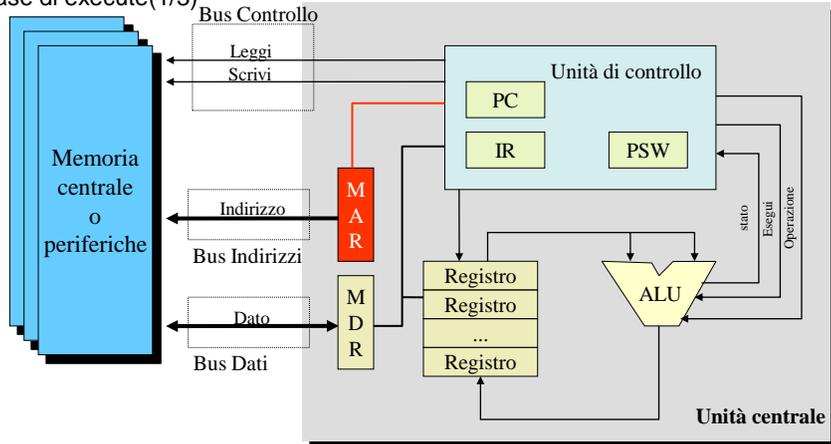


## Esempio: Lettura dalla Memoria



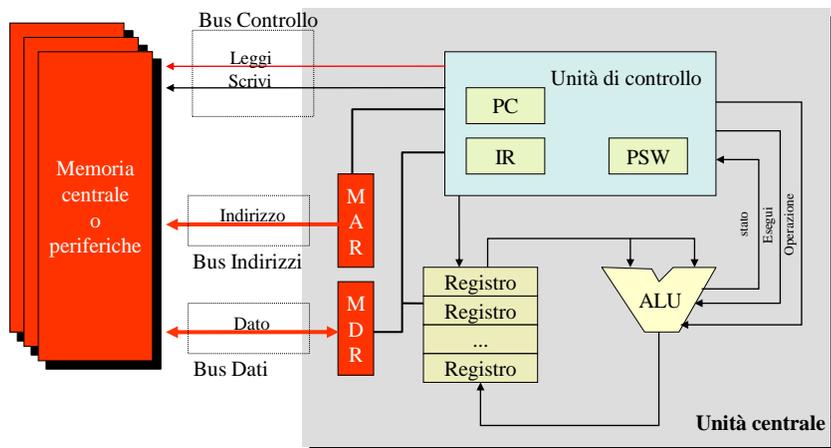
## Esempio: Lettura dalla Memoria

Fase di execute(1/3)



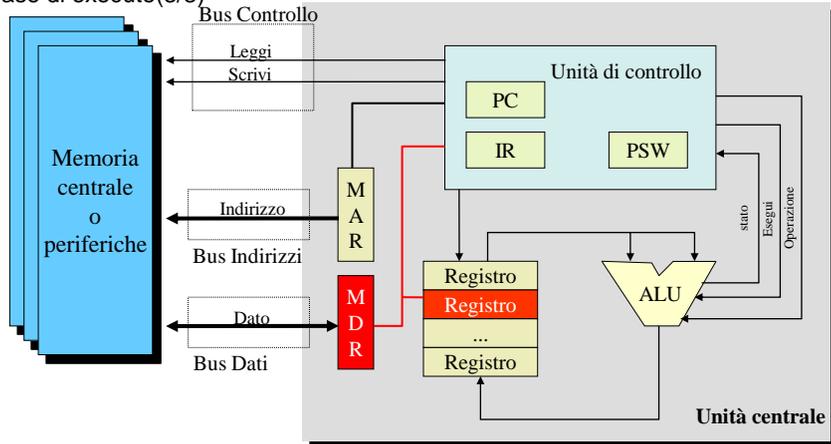
## Esempio: Lettura dalla Memoria

Fase di execute(2/3)



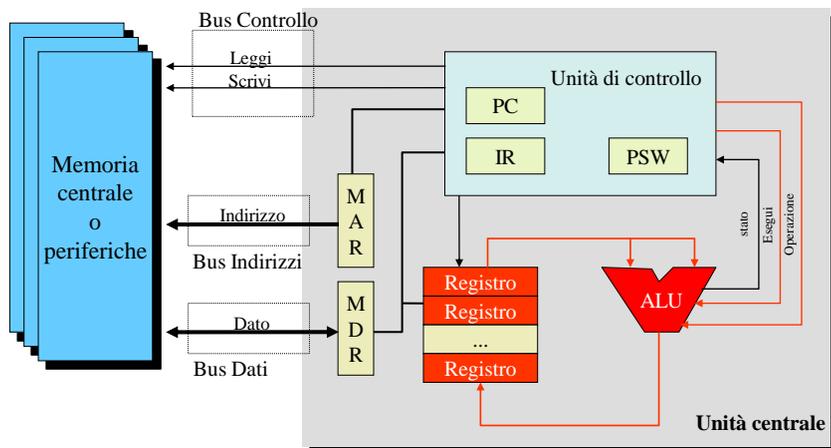
## Esempio: Lettura dalla Memoria

Fase di execute(3/3)



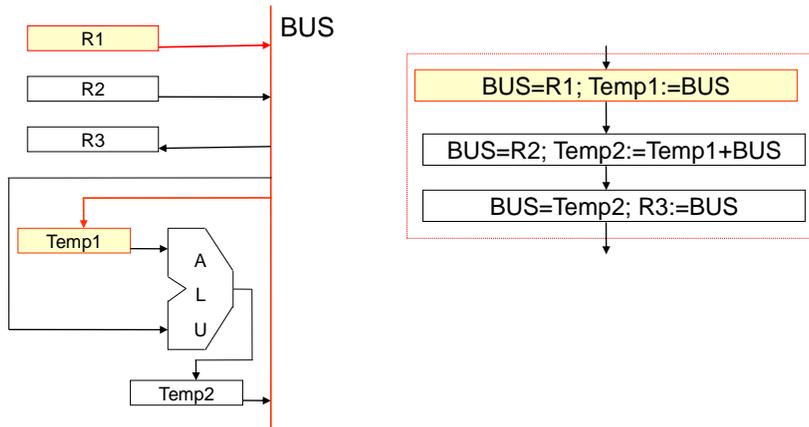
## Esempio: Somma tra 2 numeri

Fase di execute



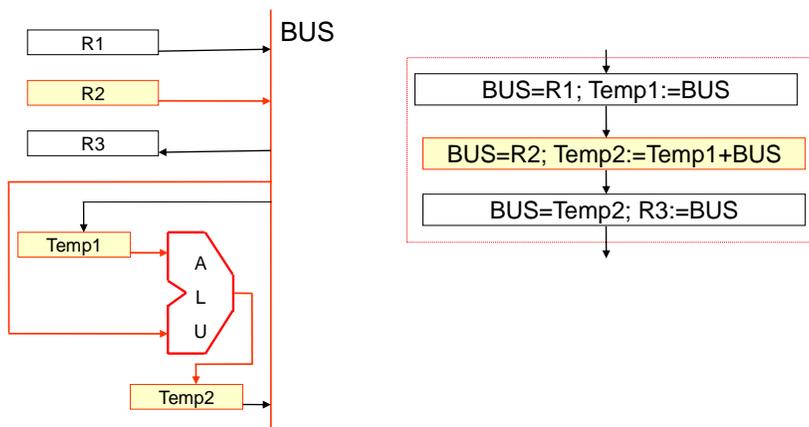
## Fase di Execute (vers.1)

Add R3, R1, R2



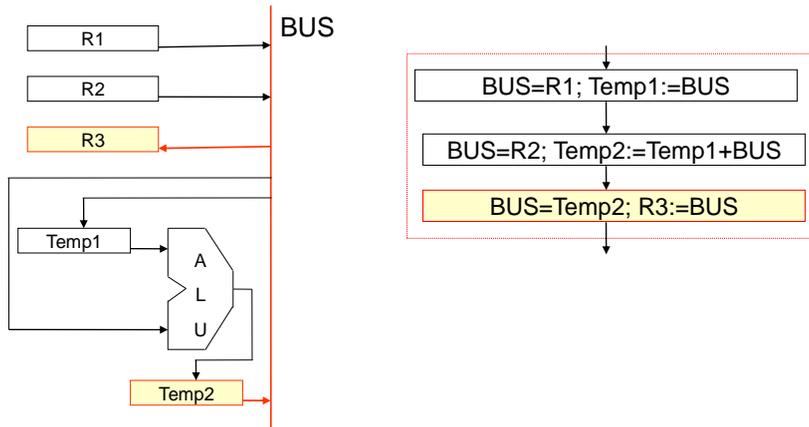
## Fase di Execute (vers.1)

Add R3, R1, R2



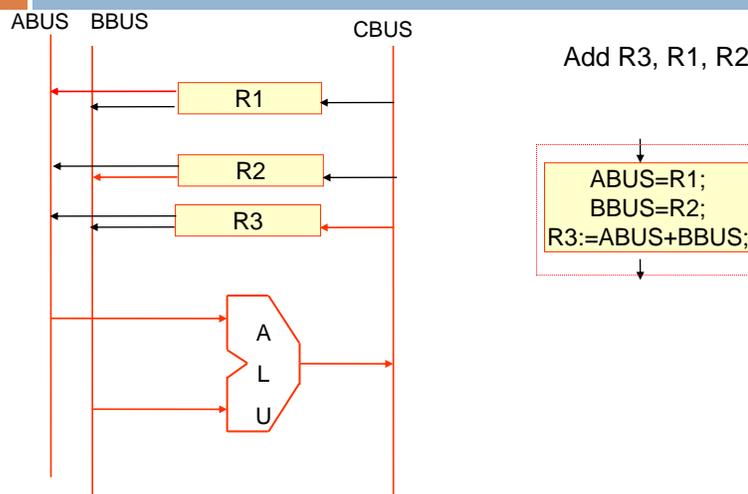
## Fase di Execute (vers.1)

Add R3, R1, R2



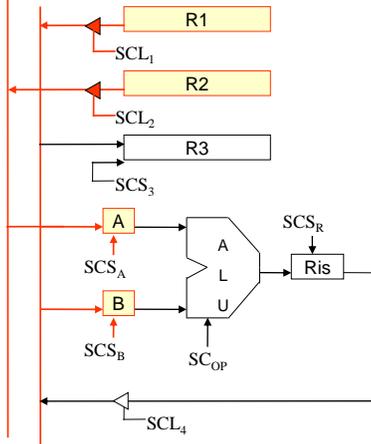
## Fase di Execute vers. 2

Add R3, R1, R2

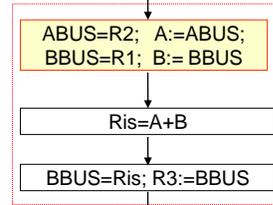


## Fase di Execute vers. 3

ABUS BBUS



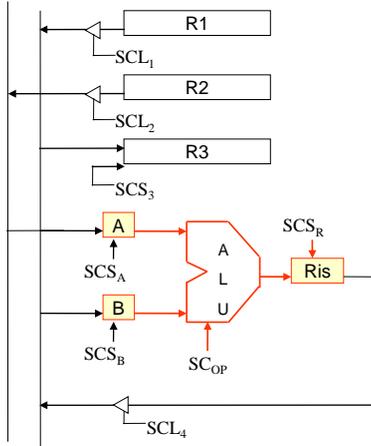
ADD R3,R1,R2



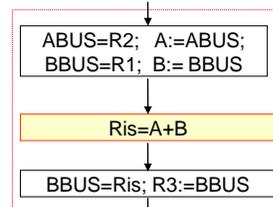
SCL<sub>1</sub>=1 → BBUS=R1  
 SCL<sub>2</sub>=1 → ABUS=R2  
 SCS<sub>A</sub>=1 → A:=ABUS  
 SCS<sub>B</sub>=1 → B:=BBUS

## Fase di Execute vers. 3

ABUS BBUS

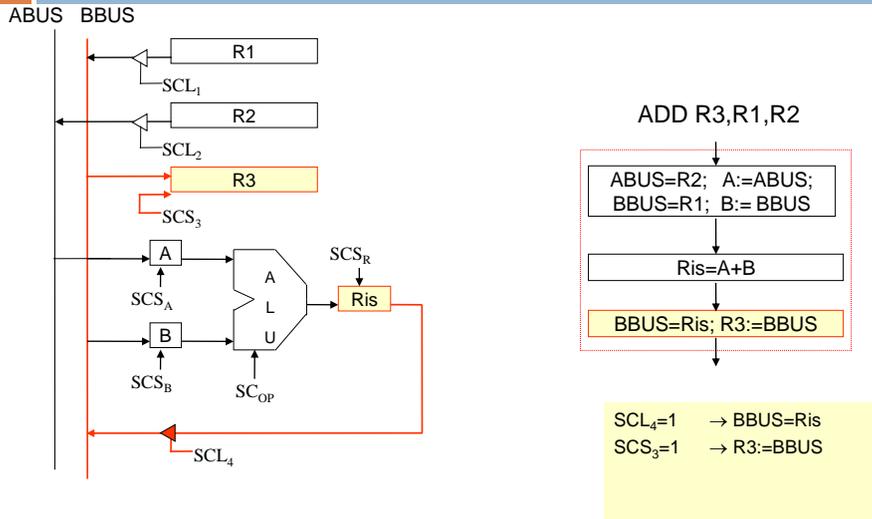


ADD R3,R1,R2



SC<sub>OP</sub>=+ → AluOutput=A+B  
 SCS<sub>R</sub>=1 → Ris:=AluOutput

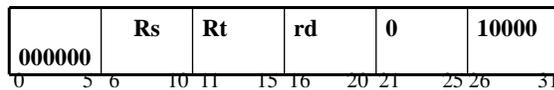
# Fase di Execute vers. 3



# Istruzioni

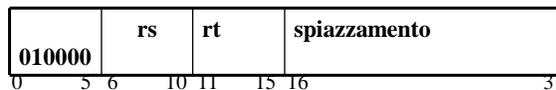
## Istruzioni di tipo R

add rd, rs, rt      Reg[rd] = Reg[rs]+Reg[rt]



## Istruzioni di salto

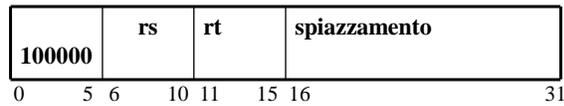
beq rs,rt spiazzamento    if(rs==rt) PC=PC+est\_segno(spiazzamento<<2)



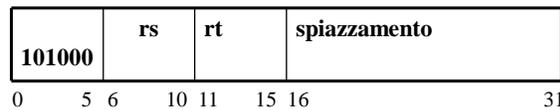
# Istruzioni

## Istruzioni di riferimento a memoria

lw rt, spiazzamento (rs)       $\text{Reg}[rt] = M[\text{Reg}[rs] + \text{est\_segno}(\text{spiazzamento})]$

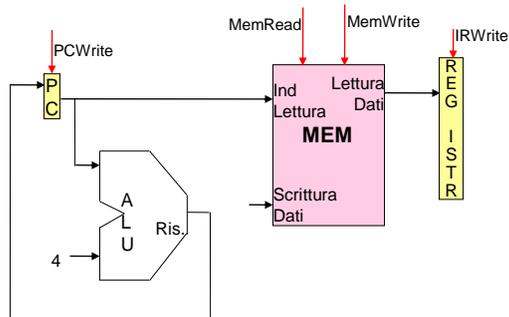


sw spiazzamento (rs, rt)       $M[\text{Reg}[rs] + \text{est\_segno}(\text{spiazzamento})] = \text{Reg}[rt]$



## Componenti per realizzare il Fetch delle istruzioni

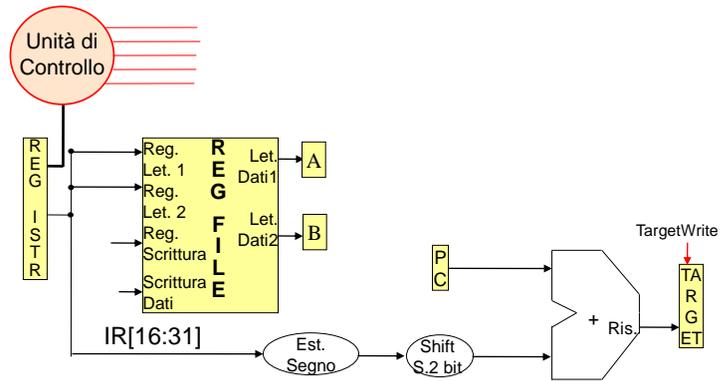
$\text{IR} := M[\text{PC}]; \text{PC} := \text{PC} + 4$



- Il contenuto del PC viene usato per indirizzare la memoria
- L'istruzione viene memorizzata nel registro IR registro istruzione

# Componenti per realizzare il Decode delle istruzioni

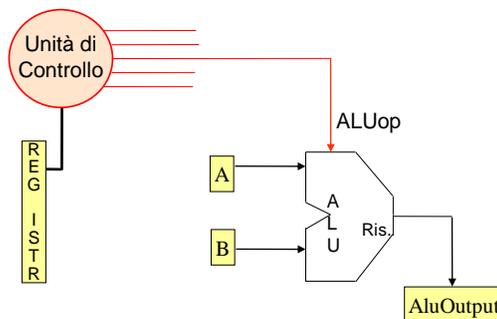
$A := \text{Reg}[ \text{IR}[6:10] ]$ ;  $B := \text{Reg}[ \text{IR}[11:15] ]$ ;  $\text{Target} := \text{PC} + \text{est\_segno}( \text{IR}[16:31] ) \ll 2$



- Il calcolo dell'indirizzo Target serve solo per le istruzioni branch

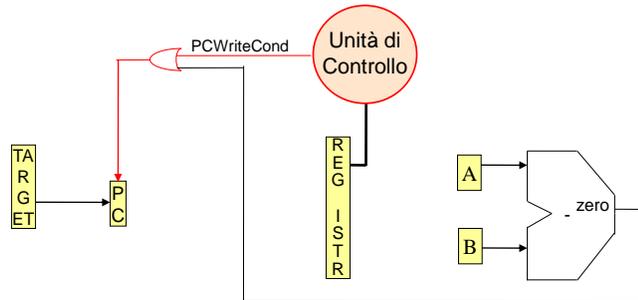
# Componenti per realizzare l'Execute delle istruzioni R

$\text{AluOutput} := A \text{ op } B$



## Componenti per realizzare l'Execute delle istruzioni Branch

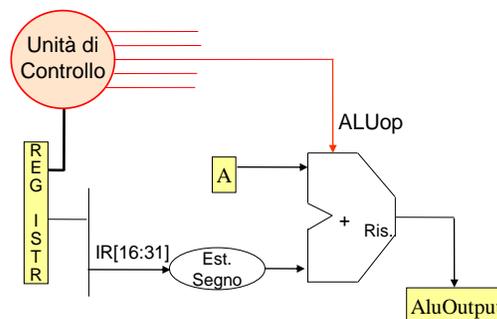
If (zero) PC:=Target



- La condizione zero è ottenuto realizzando la differenza A-B

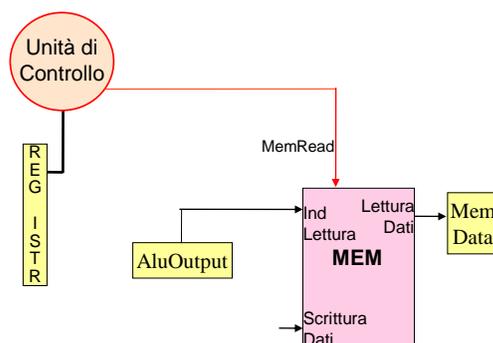
## Componenti per realizzare l'Execute delle istruzioni di accesso alla memoria

AluOutput:= A + est\_segno(IR[16:31])



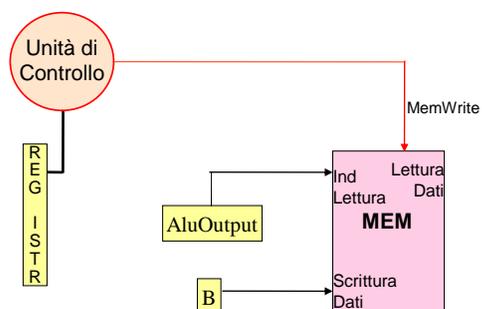
## Componenti per realizzare la lettura delle istruzioni di accesso alla memoria

$$\text{Mem-data} = M[\text{AluOutput}]$$



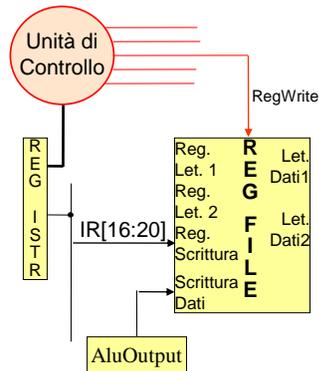
## Componenti per realizzare la scrittura delle istruzioni di accesso alla memoria

$$M[\text{AluOutput}] = B$$



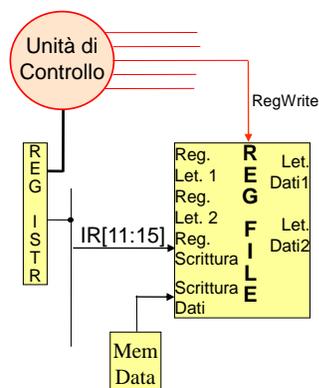
## Componenti per realizzare il Write back delle istruzioni R

Reg[IR[16:20]]:= AluOutput



## Componenti per realizzare il Write back delle istruzioni di accesso alla memoria

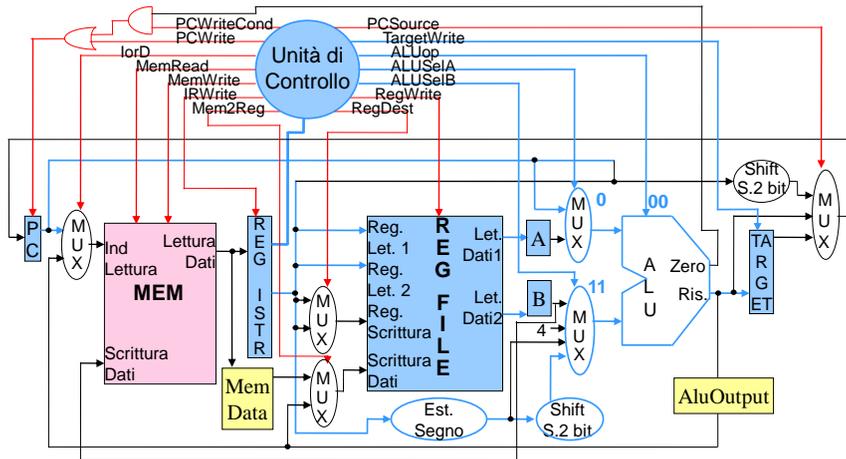
Reg[IR[11:15]]:= Mem-data





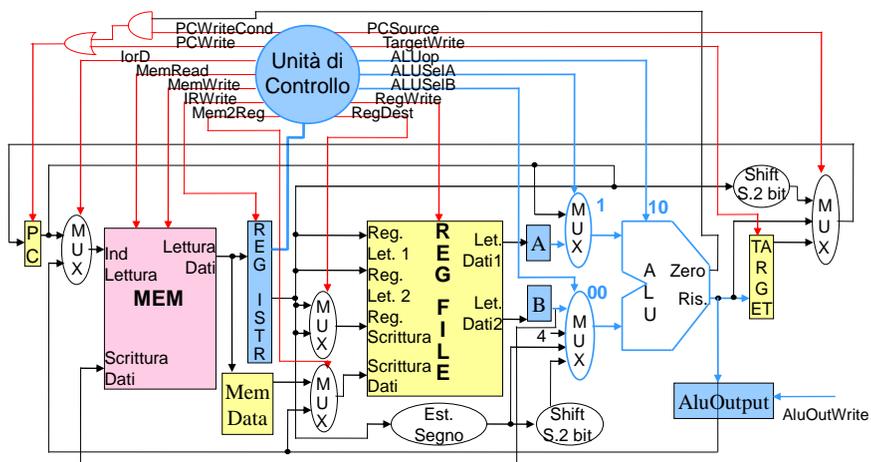
# Istruzione R: Instruction Decode

$A := \text{Reg}[ \text{IR}[6:10] ]; B := \text{Reg}[ \text{IR}[11:15] ]; \text{Target} := \text{PC} + \text{est\_segno}( \text{IR}[16:31] \ll 2 )$



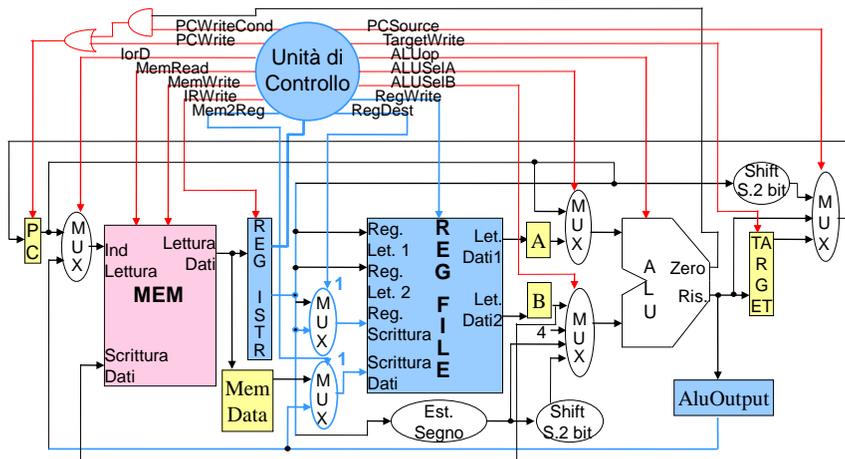
# Istruzione R: Execute

$\text{AluOutput} := A \text{ op } B$



## Istruzione R: Write back

Reg[IR[16:20]]:= AluOutput



## Esecuzione delle istruzioni di tipo R

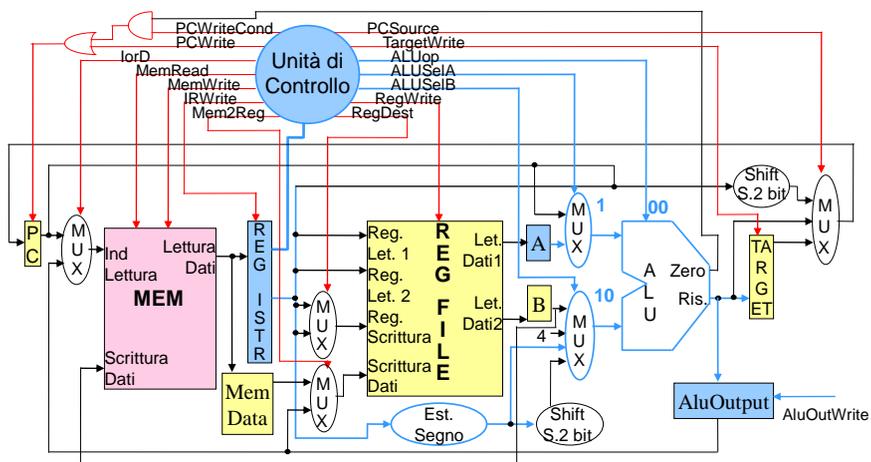
Passo	Operazioni
Prelievo dell'istruzione <i>Instruction Fetch</i>	IR:=M[PC]; PC=PC+4
Decodifica dell'istruzione <i>Instruction Decode</i>	A:=Reg[ IR[6:10] ]; B:= Reg[IR[11:15]]; Target:=PC+est_segno(IR[16:31]<<2)
Esecuzione <i>Execute</i>	AluOutput:= A op B
Scrittura <i>Write back</i>	Reg[IR[16:20]]:= AluOutput

## Istruzioni di lettura memoria

La fase di Instruction Fetch e Instruction Decode sono identiche a quelle degli altri tipi di istruzioni

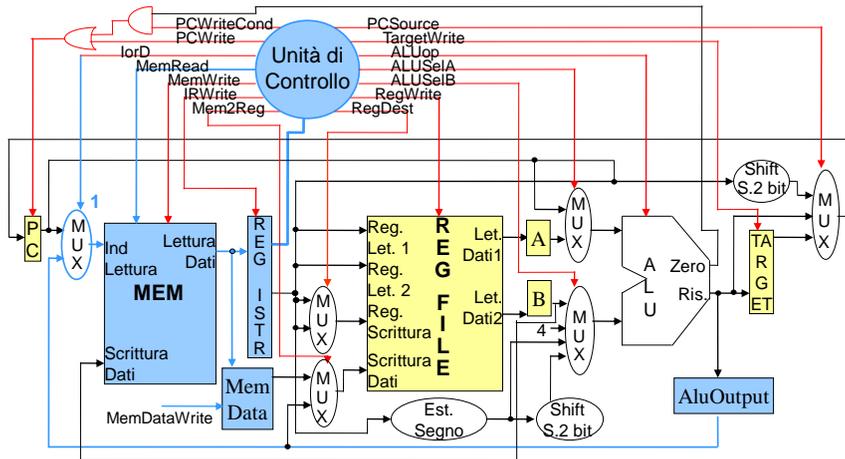
## Istruzioni di lettura memoria: Execute

$$\text{AluOutput} := A + \text{est\_segno}(\text{IR}[16:31])$$



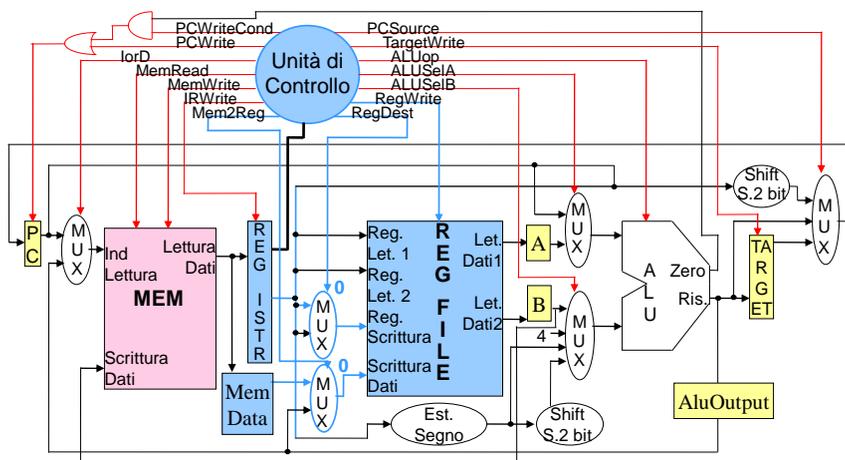
## Istruzioni di lettura memoria: Memory Access

Mem-data=M[AluOutput]



## Istruzioni di lettura memoria: Write back

Reg[IR[11:15]]:= Mem-data

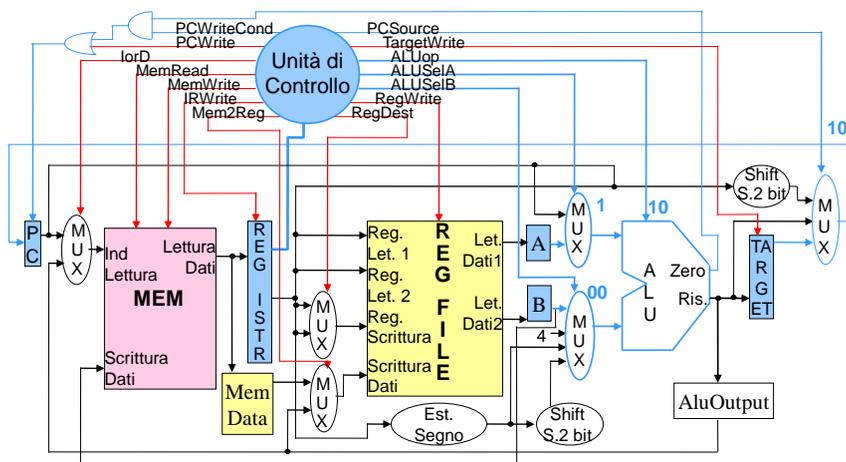


## Esecuzione delle istruzioni di lettura memoria

Passo	Operazioni
Prelievo dell'istruzione	$IR := M[PC]; PC = PC + 4$
Decodifica dell'istruzione	$A := \text{Reg}[IR[6:10]]; B := \text{Reg}[IR[11:15];$ $\text{Target} := PC + \text{est\_segno}(IR[16:31] \ll 2)$
Esecuzione	$\text{AluOutput} := A + \text{est\_segno}(IR[16:31])$
Accesso in lettura	$\text{Mem-data} = M[\text{AluOutput}]$
Scrittura	$\text{Reg}[IR[16:20]] := \text{Mem-data}$

## Istruzione branch: Execute

If (zero)  $PC := \text{Target}$

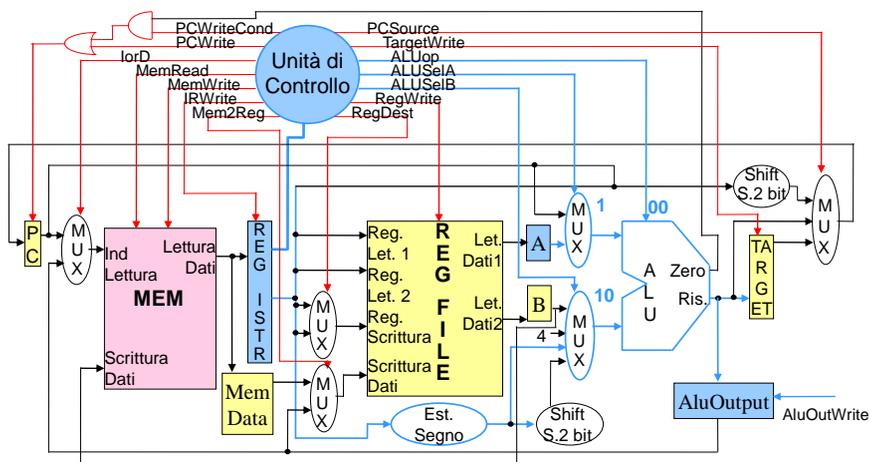


## Esecuzione delle istruzioni Branch

Passo	Operazioni
Prelievo dell'istruzione <i>Instruction Fetch</i>	$IR := M[PC]; PC = PC + 4$
Decodifica dell'istruzione <i>Instruction Decode</i>	$A := \text{Reg}[IR[6:10]]; B := \text{Reg}[IR[11:15];$ $\text{Target} := PC + \text{est\_segno}(IR[16:31] \ll 2)$
Esecuzione <i>Execute</i>	If (zero) $PC := \text{Target}$

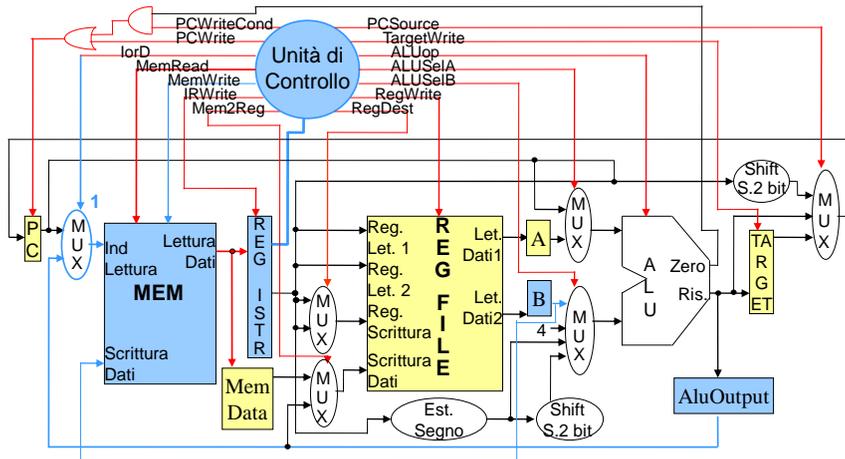
## Istruzione di scrittura memoria: Execute

$$\text{AluOutput} := A + \text{est\_segno}(IR[16:31])$$



## Istruzione di scrittura memoria: Memory access

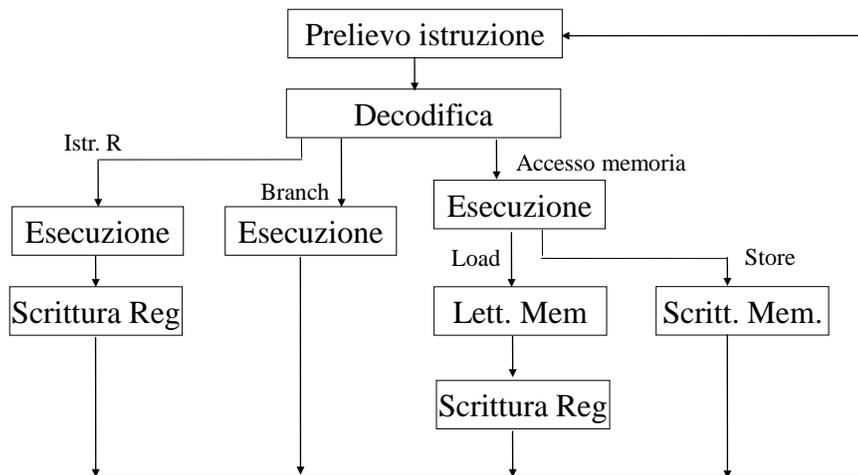
$$M[\text{AluOutput}] = B$$



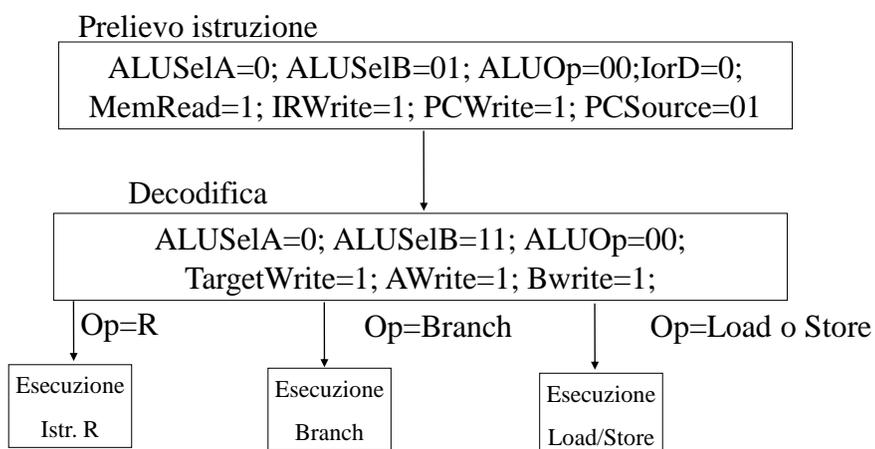
## Esecuzione delle istruzioni di scrittura memoria

Passo	Operazioni
Prelievo dell'istruzione	$IR := M[PC]; PC = PC + 4$
Decodifica dell'istruzione	$A := \text{Reg}[IR[6:10]]; B := \text{Reg}[IR[11:15]];$ $\text{Target} := PC + \text{est\_segno}(IR[16:31] \ll 2)$
Esecuzione	$\text{AluOutput} := A + \text{est\_segno}(IR[16:31])$
Accesso in scrittura	$M[\text{AluOutput}] = B;$

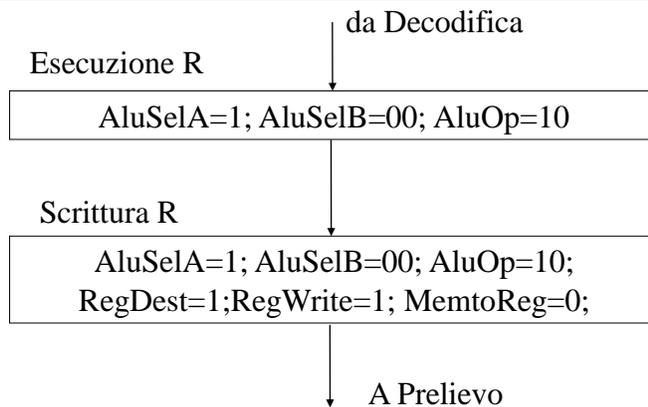
## Macchina a stati del processore



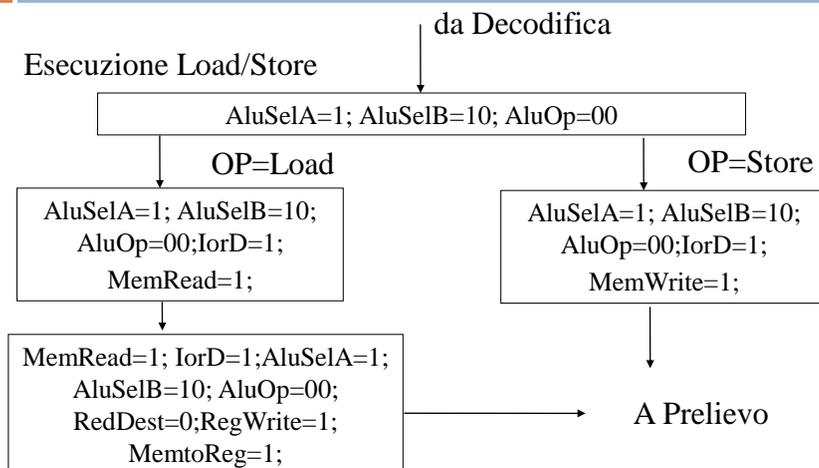
## Segnali di controllo primi due stati



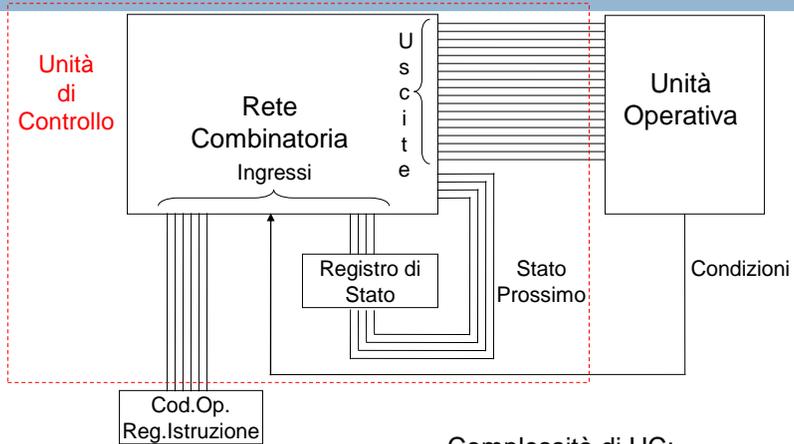
## Segnali di controllo Esecuzione Istr. R



## Segnali di controllo accesso memoria

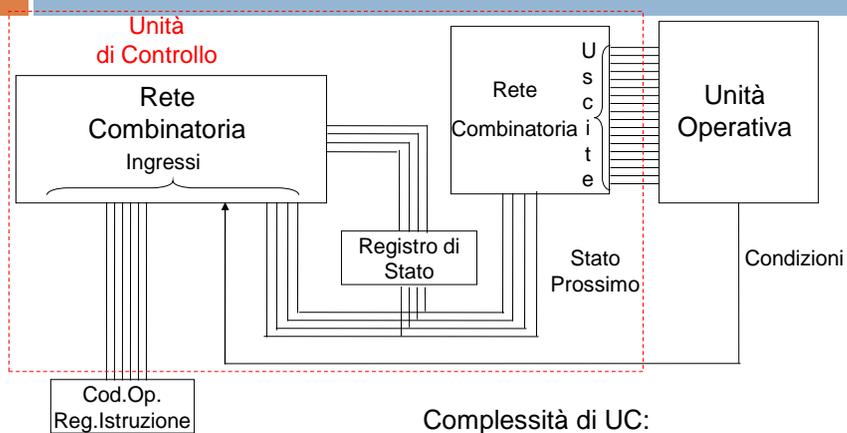


## Realizzazione cablata



Complessità di UC:  
 $2^{N_{\text{ingressi}}} \times (N_{\text{uscite}} + \log_2 N_{\text{stati}})$

## Realizzazione cablata



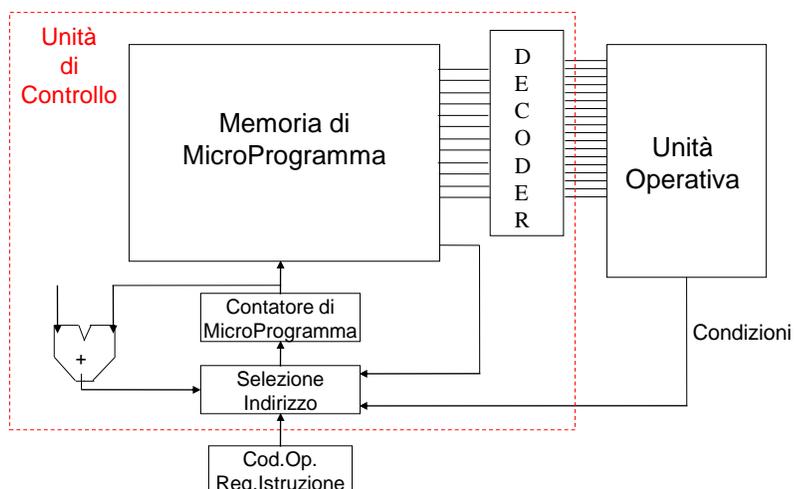
Complessità di UC:  
 $2^{N_{\text{ingressi}}} \times \log_2 N_{\text{stati}} + N_{\text{stati}} \times N_{\text{uscite}}$

# Logica cablata

- Progetto

- Come sintesi di rete sequenziale
  - ingressi: IR, Stato
  - uscite: comandi, Stato prossimo
- Uso di ROM. La rete combinatoria ha come
  - ingressi (indirizzi alla ROM): IR, stato di UO, stato di UC
  - uscite: comandi, ingressi di eccitazione dei FF di stato
- Progettazione con CAD per VLSI

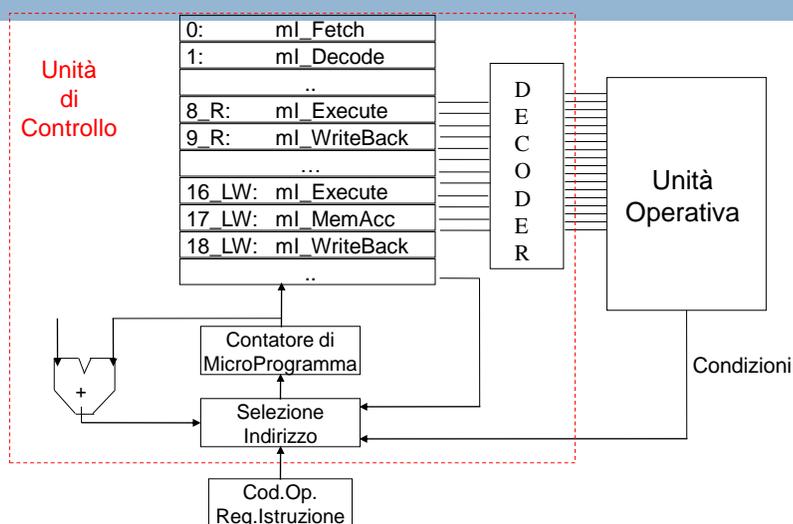
# Realizzazione microprogrammata



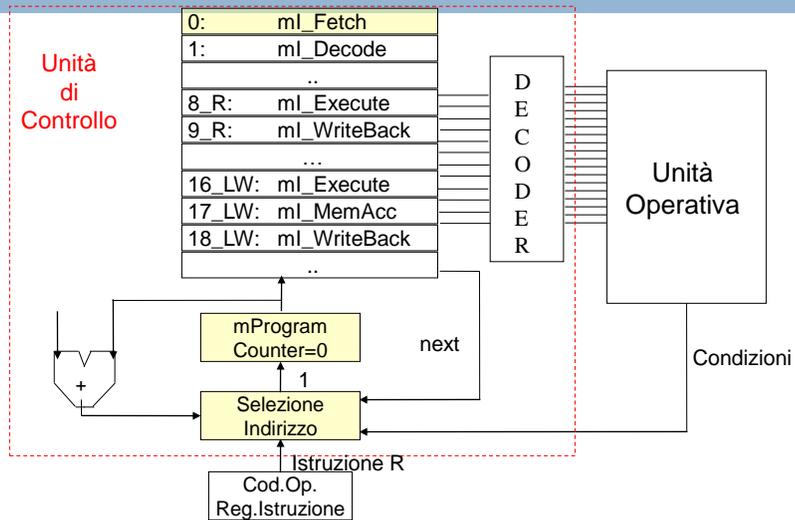
# UC microprogrammata

- Tecnica affermata negli anni 70
- UC è una sorta di calcolatore nel calcolatore
- La memoria di controllo contiene le microistruzioni:
- $\mu\text{PC}$ : contatore di microprogramma. Contiene l'indirizzo della prossima microistruzione
  - All'inizio della fase di fetch  $\mu\text{PC}$  contiene l'indirizzo ( $I_0$ ) del tratto di microprogramma corrispondente al fetch
  - Alla fine della fase di fetch e decodifica il  $\mu\text{PC}$  viene aggiornato l'indirizzo della prima microistruzione della routine che effettua le azioni richieste dalla particolare istruzione. Tale indirizzo è ottenuto a partire dal contenuto di IR
  - Al termine,  $\mu\text{PC}$  viene di nuovo caricato con ( $I_0$ )

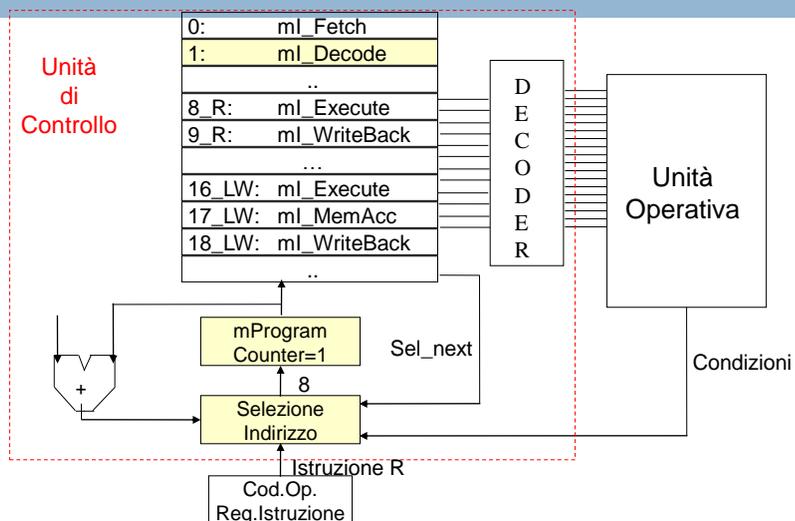
## La memoria di microprogramma



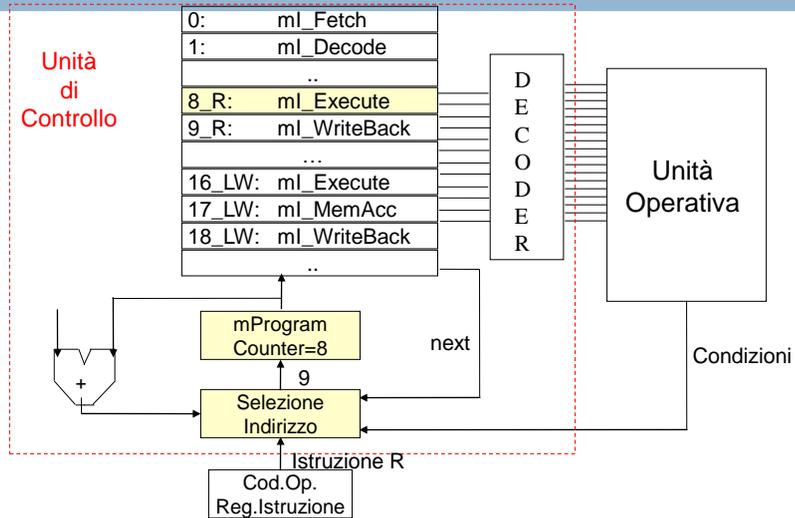
## Esecuzione delle mlstruzioni: ml\_Fetch (Istruzione R)



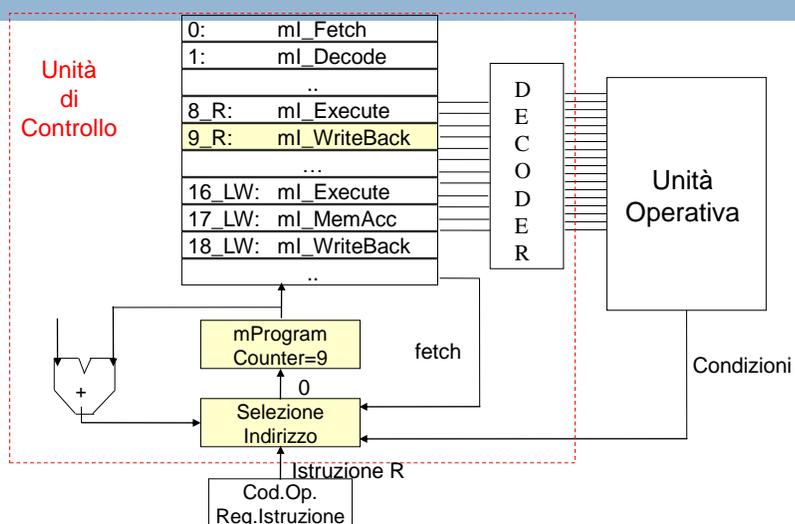
## Esecuzione delle mlstruzioni: ml\_Decode (Istruzione R)



## Esecuzione delle mlstruzioni: ml\_Execute (Istruzione R)



## Esecuzione delle mlstruzioni: ml\_WriteBack (Istruzione R)



# Memoria di Microprogramma

```
0_ml_Fetch: ALUSelA=0; ALUSelB=01; ALUOp=00;lorD=0; MemRead=1;
            IRWrite=1; PCWrite=1; PCSource=00; next;
1_ml_Decode: ALUSelA=0; ALUSelB=11; ALUOp=00; TargetWrite=1;
            Sel_next1;
.....
8_R_ml_Ex:  AluSelA=1; AluSelB=00; AluOp=10; next;
9_R_ml_WB:  AluSelA=1; AluSelB=00; AluOp=10; RegDest=1;
            RegWrite=1; MemtoReg=0; fetch;
.....
16LW_ml_Ex: AluSelA=1; AluSelB=10; AluOp=00; next;
17LW_ml_MM: AluSelA=1; AluSelB=10; AluOp=00;lorD=1;
            MemRead=1;next;
18LW_ml_WB: MemRead=1; lorD=1;AluSelA=1; AluSelB=10; AluOp=00;
            RedDest=0;RegWrite=1; MemtoReg=1; fetch;
.....
```

# Microprogrammazione

- Aniché indicare in modo esplicito il valore dei segnali di controllo per ogni istruzione, possiamo pensare le microistruzione composte da  $\mu$ operazioni rappresentate in modo simbolico
- Es. La  $\mu$ istruzione di fetch in corrispondenza della quale i segnali di controllo hanno i seguenti valori

```
ALUSelA=0; ALUSelB=01 ; ALUOp=00; lorD=0; MemRead=1;
IRWrite=1; PCWrite=1; PCSource=00;
```

può essere vista come l'insieme delle seguenti  $\mu$ operazioni

```
Add(PC,4); Lettura(PC); Scrivi(IR); Scrivi(PC); PCSource=PC;
```

## Controllo microprogrammato

0\_ml\_Fetch: Add(PC,4); Lettura(PC); Scrivi(IR); Scrivi(PC);  
PCSource=PC; next;

1\_ml\_DeCode: Add( PC, Est\_segno); Scrivi(Target); Sel\_next1;

8\_R\_ml\_Ex: func(rs,rt); Scrivi(AluOutput);next;

9\_R\_ml\_WB: Scrivi(rd); fetch;

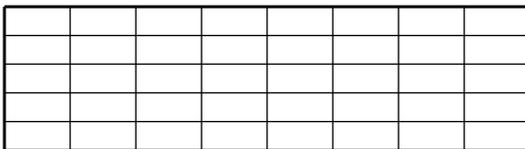
16LW\_ml\_Ex: add(rs, Est\_segno); Sel\_next2;

17LW\_ml\_MM: add(rs,Est\_segno); Lettura(ALU); next;

18LW\_ml\_WB: add(rs,Est\_segno); Lettura(ALU); Scrivi(rt); next;; fetch;

32BR\_ml\_Ex: ...; ....; fetch;

## Microprogrammazione orizzontale

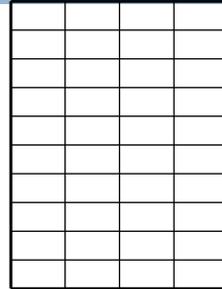


Orizzontale

- Nella microprogrammazione orizzontale le microistruzioni contengono molte microoperazioni eseguite in parallelo
- Il microprogramma risulta costituito da un numero limitato di microistruzioni
- **Vantaggi:** buona velocità nella esecuzione delle istruzione
- **Svantaggi:** notevole spreco di memoria

## Microprogrammazione verticale

- Nella microprogrammazione verticale le microistruzioni contengono poche microoperazioni eseguite in parallelo
- Il microprogramma risulta costituito da un elevato numero di microistruzioni
- **Svantaggi:** ridotta velocità nella esecuzione delle istruzioni
- **Vantaggi:** buon uso della memoria



## Cablata o microprogrammata?

- Fino a fine anni '60: logica cablata (PDP8, HP 2116)
- Anni '70: microprogrammazione (VAX, Z80, 8086, 68000)
  - Repertorio di istruzioni molto esteso e variato: **CISC**
  - Il VAX 11/789 (Digital) e il 370/168 (IBM) avevano oltre 400.000 bit di memoria di controllo
- Dagli anni '80 si è tornati alla logica cablata;
  - Affermazione delle macchine **RISC**
- Istruttivo è esaminare l'evoluzione dell'architettura Intel: da CISC a (praticamente) RISC