

Corso di Architettura degli Elaboratori e Laboratorio (M-Z)

# Struttura base del processore

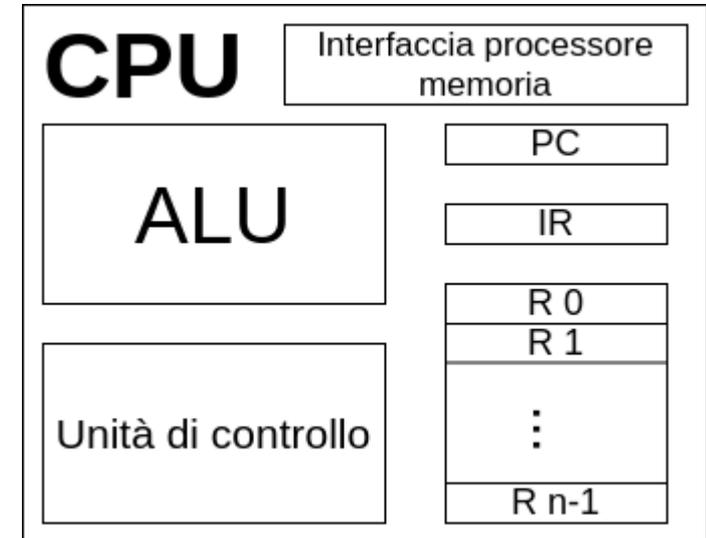
*Nino Cauli*



UNIVERSITÀ  
degli STUDI  
di CATANIA

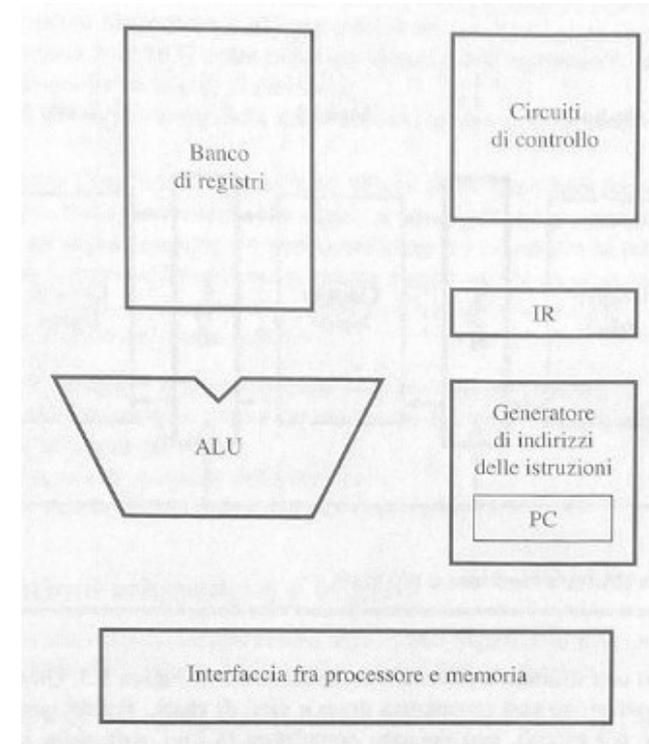
Dipartimento di Matematica e Informatica

- È un **CIRCUITO ELETTRONICO INTEGRATO** (chip) con il ruolo di **CERVELLO** del calcolatore
- Capace di caricare ed eseguire le **ISTRUZIONI ELEMENTARI** necessarie per eseguire i **PROGRAMMI**
- Esempi di istruzioni elementari: operazioni aritmetiche, operazioni logiche, confronti, salti incondizionati e condizionati.

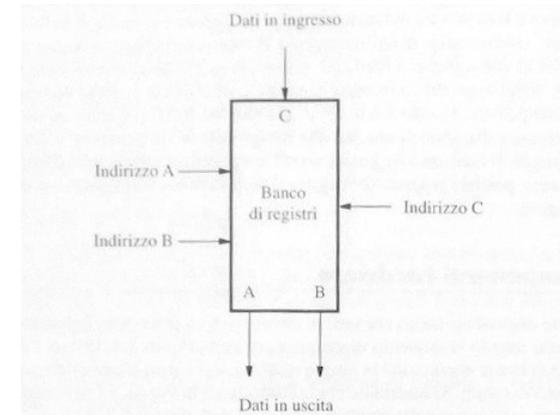


# Processore (CPU)

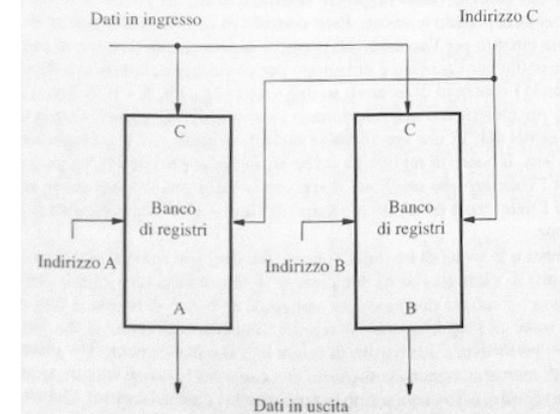
- L'**UNITÀ ARITMETICA-LOGICA (ALU)** esegue le operazioni aritmetiche e logiche necessarie ad eseguire le istruzioni
- **CIRCUITI DI CONTROLLO** generano i bit di controllo per gestire il funzionamento della CPU
- **BANCO DI REGISTRI**: blocco di memoria contenente i registri generici della CPU
- **PC** e **IR**: registri che contengono rispettivamente l'indirizzo della prossima istruzione e l'istruzione in esecuzione
- **GENERATORE DI INDIRIZZI**: aggiorna il contenuto di PC
- **INTERFACCIA PROCESSORE MEMORIA** gestisce il trasferimento di dei dati tra memoria e CPU



- Blocco di memoria piccolo e veloce
- Consiste in vari registri con un circuito per l'accesso in scrittura e lettura
- Lettura contemporanea di 2 registri (porte di uscita A e B) e scrittura di un singolo registro (porta di ingresso C)
- Due ingressi per gli indirizzi di lettura (Indirizzo A, Indirizzo B) e uno per l'indirizzo di scrittura (Indirizzo C)
- Per realizzare la lettura simultanea di due registri esistono due metodi:
  - Singolo banco di registri con percorsi dati e circuiti di accesso duplicati
  - Due copie del banco di registri, una per A e l'altra per B

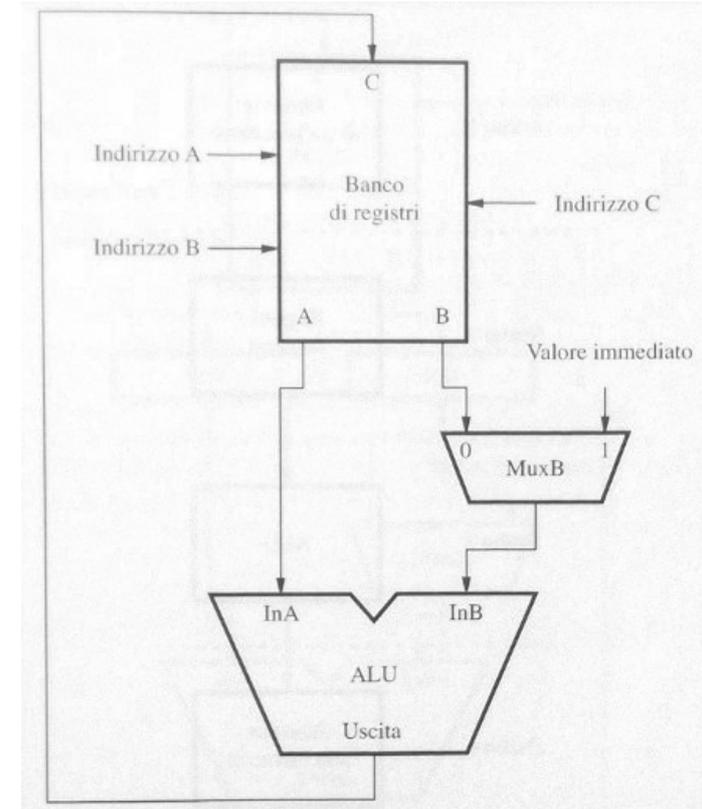


(a) Blocco di memoria singolo



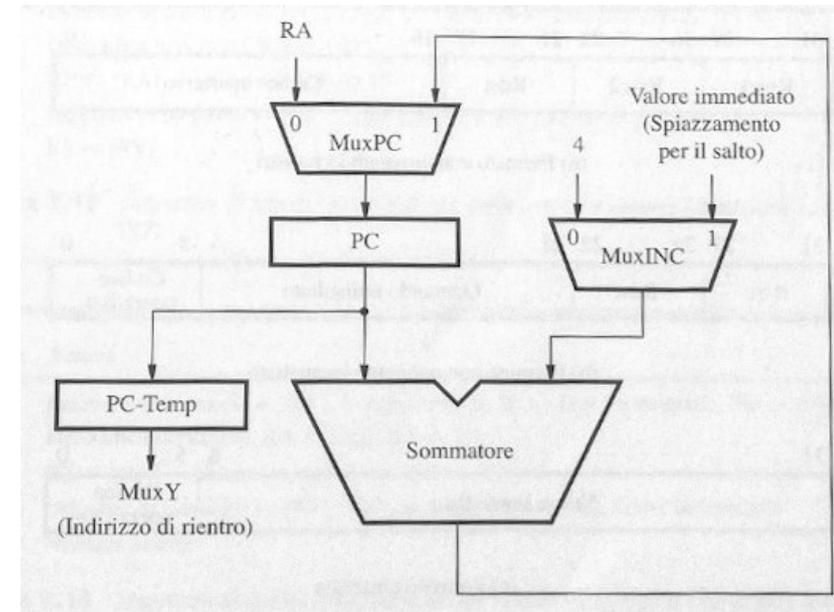
(b) Due blocchi di memoria

- Esegue le operazioni aritmetiche e logiche quali somma, sottrazione, AND, OR, XOR, etc.
- 2 porte di input (InA e InB) rappresentanti gli operandi i ingresso
- 1 uscita contenente il risultato dell'operazione
- Un collegamento semplificato tra ALU e Banco di registri è mostrato in figura
- Gli ingressi dell'ALU sono collegati ai registri di uscita A e B
- Un MUX è usato per scegliere se passare un valore immediato come secondo operando



# Generatore di indirizzi delle istruzioni

- Circuito usato per generare l'indirizzo della prossima istruzione da inserire nel PC
- Un sommatore è usato sia per incrementare PC di una parola (4 byte) e sia per sommargli uno spiazzamento nel caso di salto
- MuxINC seleziona che tipo di incremento effettuare
- MuxPC seleziona se aggiornare PC con l'incremento calcolato o con un indirizzo specifico (chiamata a sottoprogramma)
- Il registro temporaneo PC-Temp è usato per salvare il valore di PC da inserire nel LR durante una chiamata a sottoprogramma



- Per eseguire un'istruzione, il processore deve eseguire i seguenti 3 passi:
  1. Prelievo dell'istruzione dalla memoria puntata da PC:  $IR \leftarrow [[PC]]$
  2. Incremento di PC di 4 unità (prossima istruzione):  $PC \leftarrow [PC] + 4$
  3. Esecuzione dell'istruzione prelevata
- I primi due passi vengono chiamati **fase di prelievo (FETCH PHASE)**
- Il terzo passo è chiamato **fase di esecuzione (EXECUTION PHASE)**
- Durante la fase di esecuzione si possono svolgere diverse azioni: **lettura/scrittura da/in una locazione di memoria, lettura da registri, esecuzione di operazioni aritmetiche e logiche, etc.**

## Load R5, X(R7)

1. Prelievo dell'istruzione ed incremento del PC
2. Decodifica dell'istruzione e lettura del contenuto del registro R7
3. Calcolo dell'indirizzo effettivo
4. Lettura dell'operando sorgente dalla memoria
5. Caricamento dell'operando nel registro di destinazione R5

## Add R3, R4, R5

1. Prelievo dell'istruzione ed incremento del PC
2. Decodifica dell'istruzione e lettura dei contenuti dei registri sorgenti R4 e R5
3. Calcolo della somma  $[R4] + [R5]$
4. Caricamento del risultato nel registro di destinazione R3

## Store R6, X(R8)

1. Prelievo dell'istruzione ed incremento del PC
2. Decodifica dell'istruzione e lettura dei registri R6 e R8
3. Calcolo dell'indirizzo effettivo  $X + [R8]$
4. Immagazzinamento del contenuto di R6 nella locazione di memoria  $X + [R8]$

## Load R5, X(R7)

1. Prelievo dell'istruzione ed incremento del PC
2. Decodifica dell'istruzione e lettura del contenuto del registro R7
3. Calcolo dell'indirizzo effettivo
4. Lettura dell'operando sorgente dalla memoria
5. Caricamento dell'operando nel registro di destinazione R5

## Add R3, R4, R5

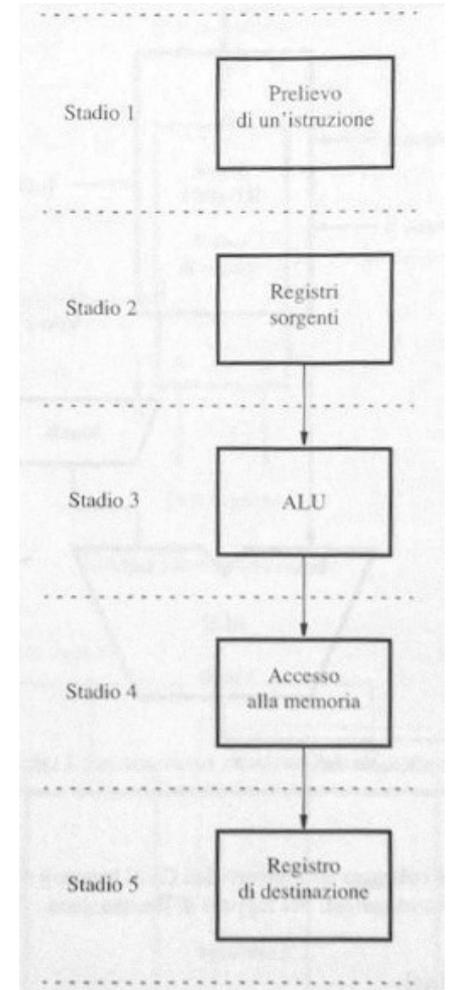
1. Prelievo dell'istruzione ed incremento del PC
2. Decodifica dell'istruzione e lettura dei contenuti dei registri sorgenti R4 e R5
3. Calcolo della somma  $[R4] + [R5]$
4. **Nessuna azione**
5. Caricamento del risultato nel registro di destinazione R3

## Store R6, X(R8)

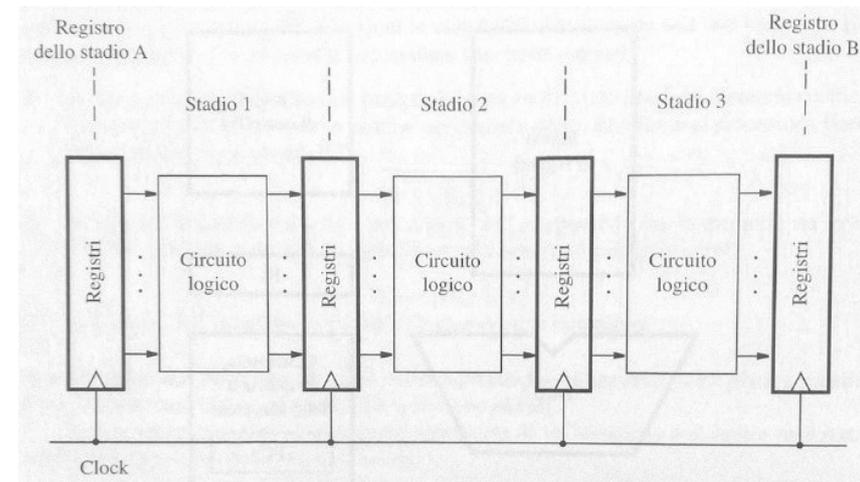
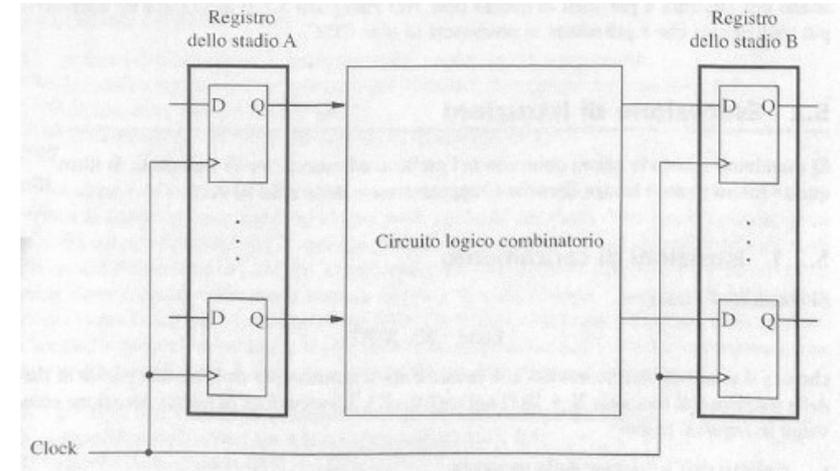
1. Prelievo dell'istruzione ed incremento del PC
2. Decodifica dell'istruzione e lettura dei registri R6 e R8
3. Calcolo dell'indirizzo effettivo  $X + [R8]$
4. Immagazzinamento del contenuto di R6 nella locazione di memoria  $X + [R8]$
5. **Nessuna azione**

# Organizzazione a cinque stadi

- Tutte le operazioni possono essere eseguite in 5 stadi distinti (saltandone alcuni nel caso sia necessario):
  1. Preleva un'istruzione e incrementa il contatore del programma
  2. Decodifica l'istruzione e legge i registri dal banco dei registri
  3. Esegui un'operazione dell'ALU
  4. Leggi o scrivi dati in memoria
  5. Scrivi il risultato nel registro di destinazione

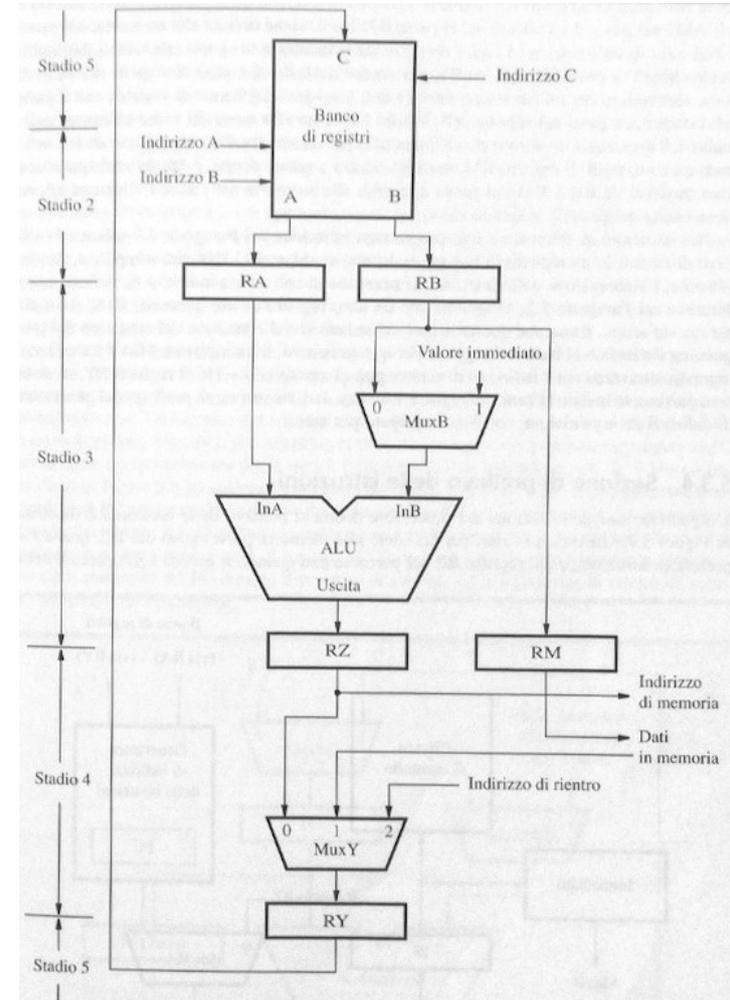


- Per temporizzare il trasferimento di dati nel circuito del processore si usa un segnale di clock
- Se volessimo calcolare il risultato in un solo stadio, un ciclo di clock dovrebbe essere lungo a sufficienza per effettuare tutti i passi dell'istruzione
- Possiamo dividere il circuito combinatorio in più circuiti più semplici in cascata (uno per stadio), inserendo registri temporanei tra uno stadio e l'altro
- Questa struttura si adatta a funzionare in pipeline



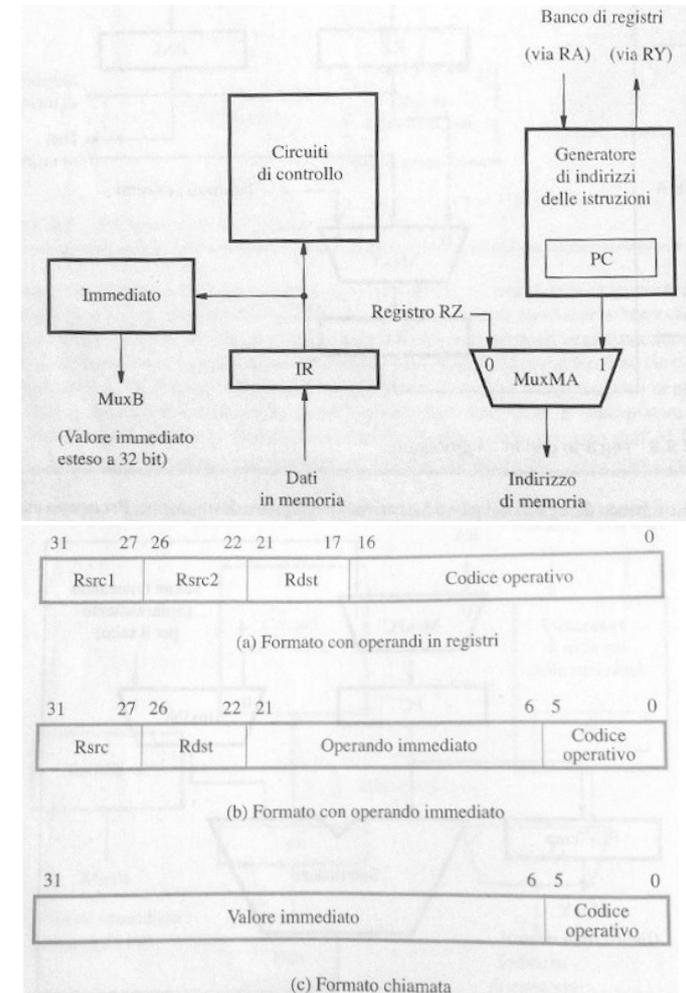
# Datapath (percorso dati)

- **Stadio 2:**
  - Le porte di uscita A e B del banco di registri vengono copiate nei registri temporanei RA e RB
- **Stadio 3:**
  - RA viene usato come primo ingresso dell'ALU
  - MuxB sceglie RB o un valore immediato come secondo ingresso
  - Il risultato dell'ALU viene copiato in RZ
  - RB viene copiato in RM
- **Stadio 4:**
  - Se necessario l'indirizzo in RZ viene mandato all'interfaccia processore memoria
  - Se necessario i dati in RM vengono salvati in memoria
  - MuxY sceglie se salvare su RY il risultato di un'operazione, dei dati della memoria o l'indirizzo di rientro da sottoprogramma
- **Stadio 5:**
  - Il contenuto di RY viene salvato nel Banco dei registri



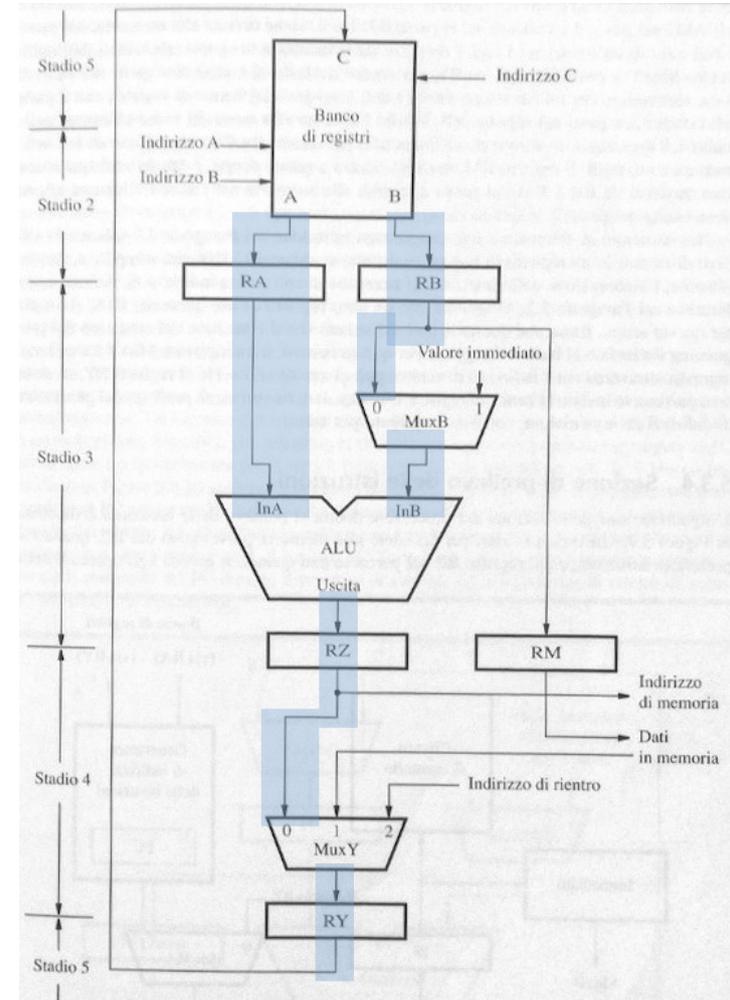
# Prelievo delle istruzioni (Stadio 1)

- Per prelevare un'istruzione dalla memoria, MuxMA sceglie se mandare l'indirizzo in RZ o quello in PC all'interfaccia processore/memoria
- L'istruzione da caricare è salvata nel registro IR
- L'istruzione viene decodificata dai circuiti di controllo e i bit corrispondenti al valore immediato vengono passati a MuxB



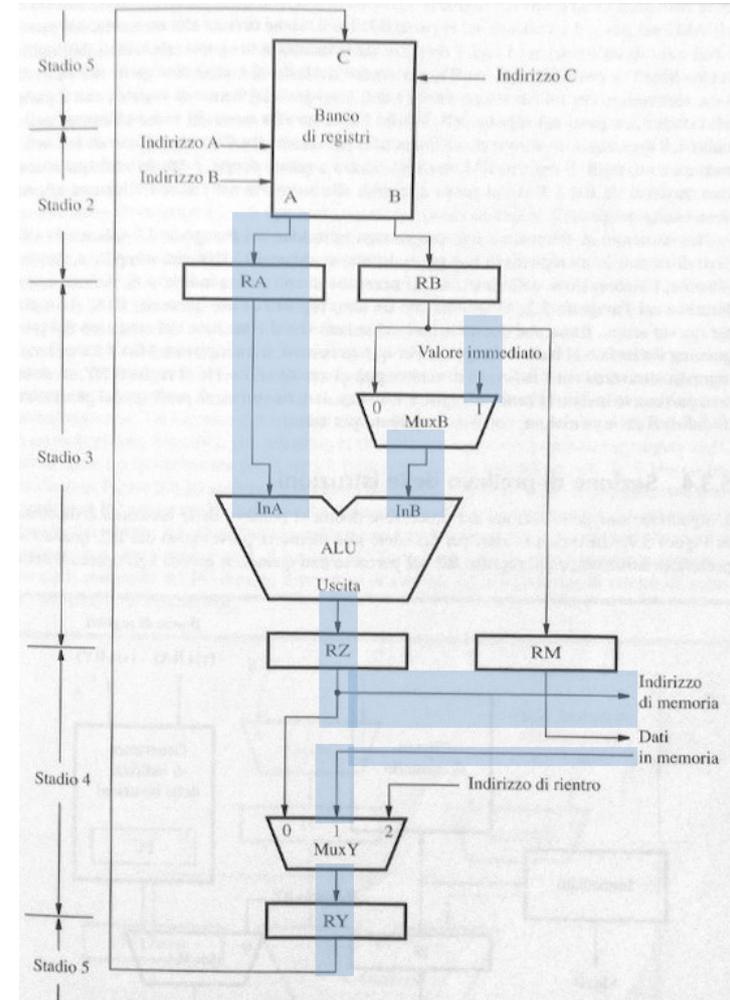
## Add R3, R4, R5

1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
2. Decodifica istruzione, RA  $\leftarrow$  [R4], RB  $\leftarrow$  [R5]
3. RZ  $\leftarrow$  [RA] + [RB]
4. RY  $\leftarrow$  [RZ]
5. R3  $\leftarrow$  [RY]



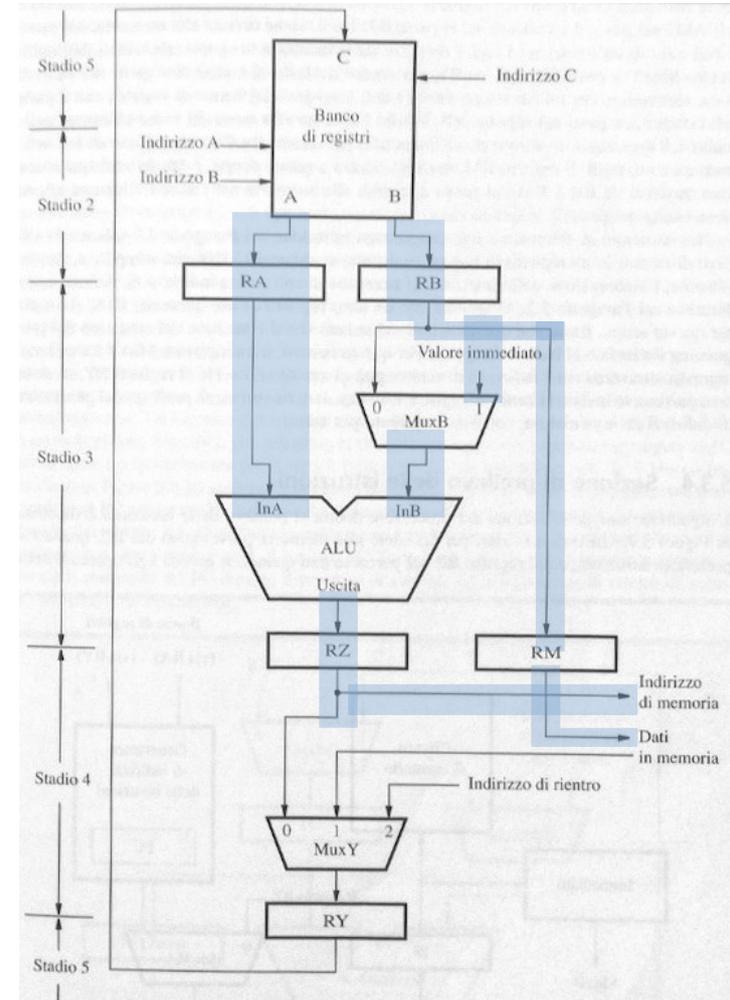
## Load R5, X(R7)

1. Indirizzo di memoria  $\leftarrow [PC]$ , Leggi memoria,  $IR \leftarrow$  Dati da memoria,  $PC \leftarrow [PC] + 4$
2. Decodifica istruzione,  $RA \leftarrow [R7]$
3.  $RZ \leftarrow [RA] + X$
4. Indirizzo di memoria  $\leftarrow [RZ]$ , Leggi memoria,  $RY \leftarrow$  Dati memoria
5.  $R5 \leftarrow [RY]$



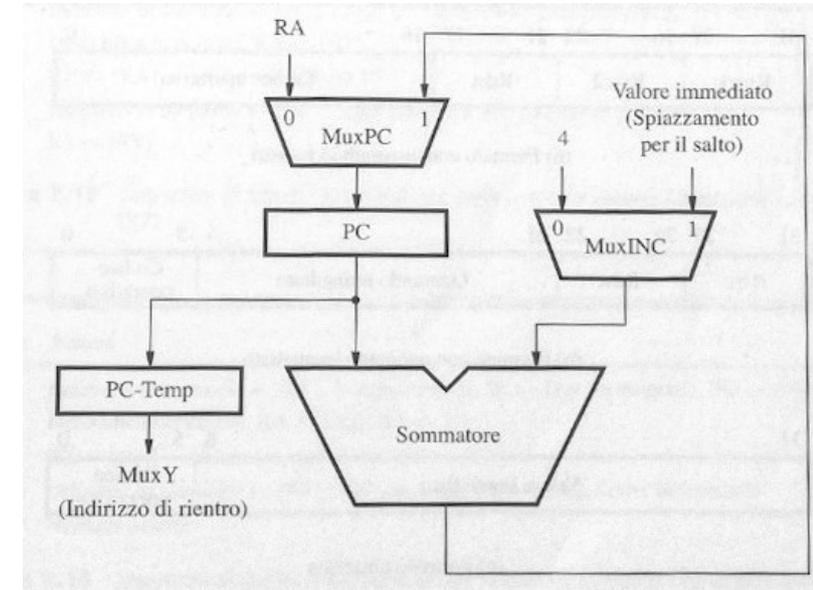
## Store R6, X(R8)

1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
2. Decodifica istruzione, RA  $\leftarrow$  [R8], RB  $\leftarrow$  [R6]
3. RZ  $\leftarrow$  [RA] + X, RM  $\leftarrow$  [RB]
4. Indirizzo di memoria  $\leftarrow$  [RZ], Dati per memoria  $\leftarrow$  [RM], scrivi memoria
5. Nessuna azione



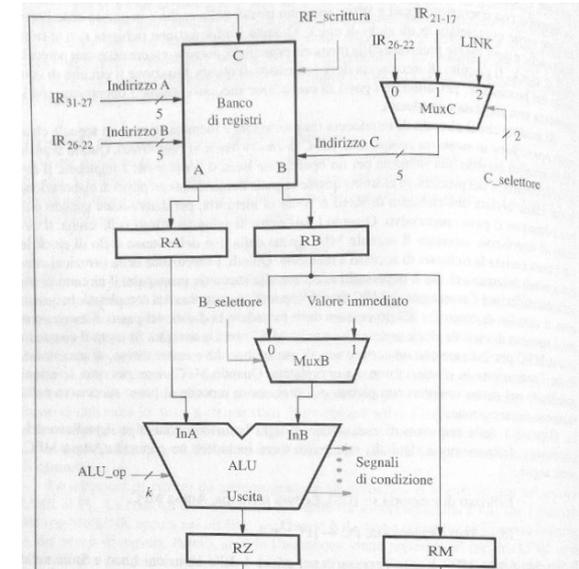
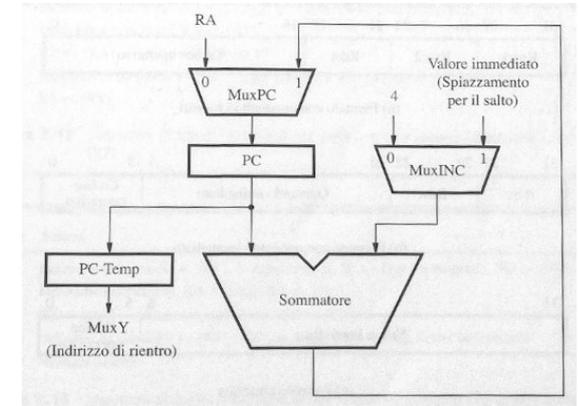
## Branch CICLO

1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
2. Decodifica istruzione
3. PC  $\leftarrow$  [PC] + Spiazzamento salto
4. Nessuna azione
5. Nessuna azione



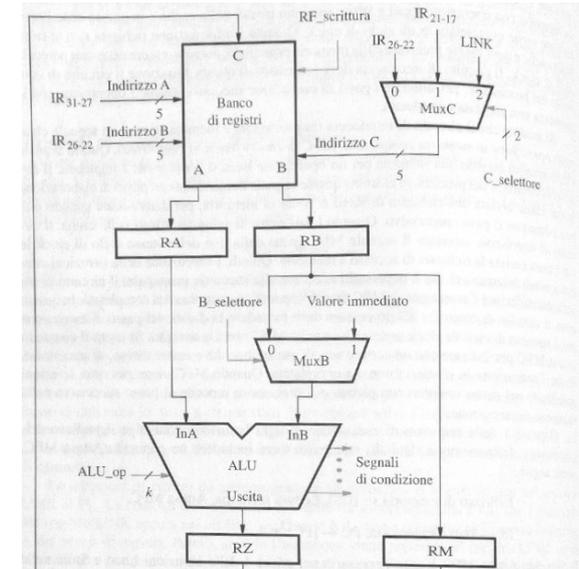
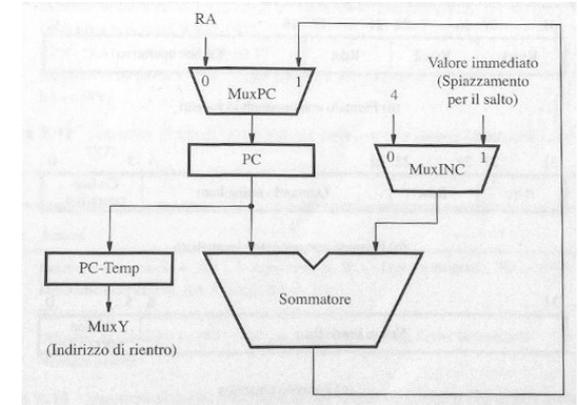
## Branch\_if\_[R5]=[R6] CICLO

1. Indirizzo di memoria  $\leftarrow [PC]$ , Leggi memoria, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow [PC] + 4$
2. Decodifica istruzione, RA  $\leftarrow [R5]$ , RB  $\leftarrow [R6]$
3. Confronta [RA] e [RB], Se [RA] = [RB] allora PC  $\leftarrow [PC] +$  Spiazzamento salto
4. Nessuna azione
5. Nessuna azione



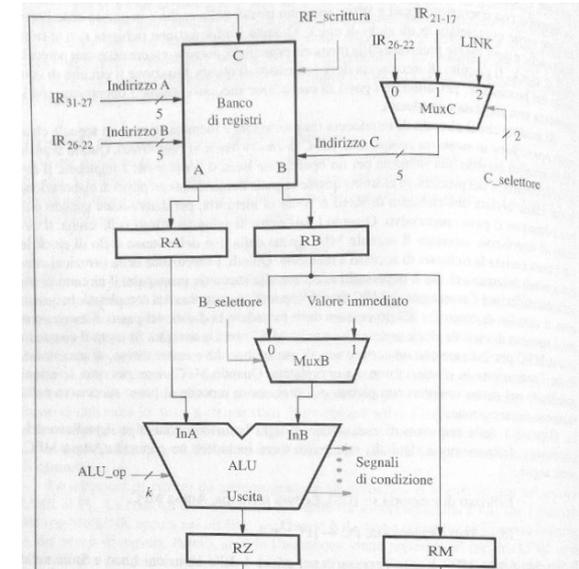
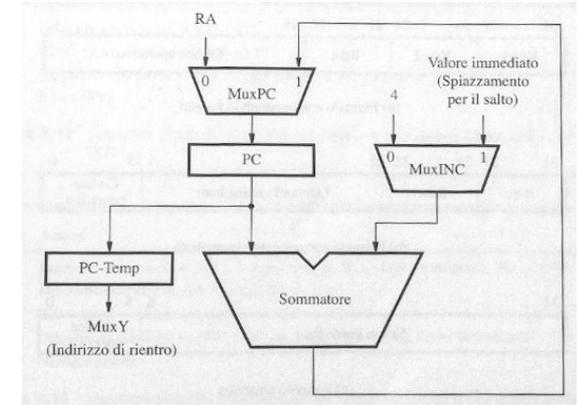
## Call SUB1

1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
2. Decodifica istruzione
3. PC-Temp  $\leftarrow$  [PC], PC  $\leftarrow$  [PC] + Spiazzamento salto
4. RY  $\leftarrow$  [PC-Temp]
5. RL  $\leftarrow$  [RY]



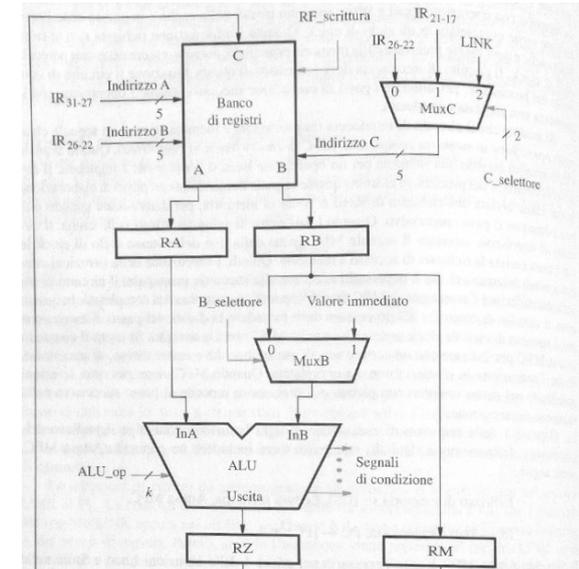
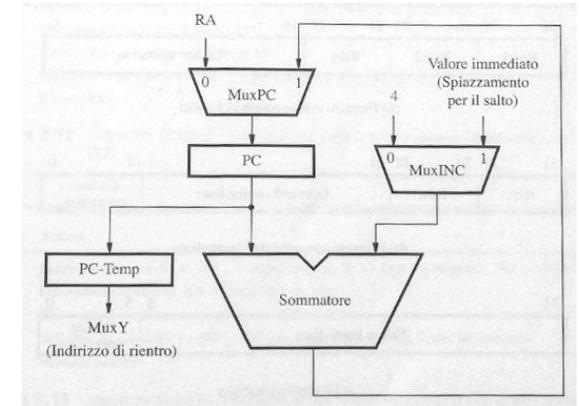
## Return

1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
2. Decodifica istruzione, RA  $\leftarrow$  [LR]
3. PC  $\leftarrow$  [RA]
4. Nessuna azione
5. Nessuna azione



## Call\_Register R9

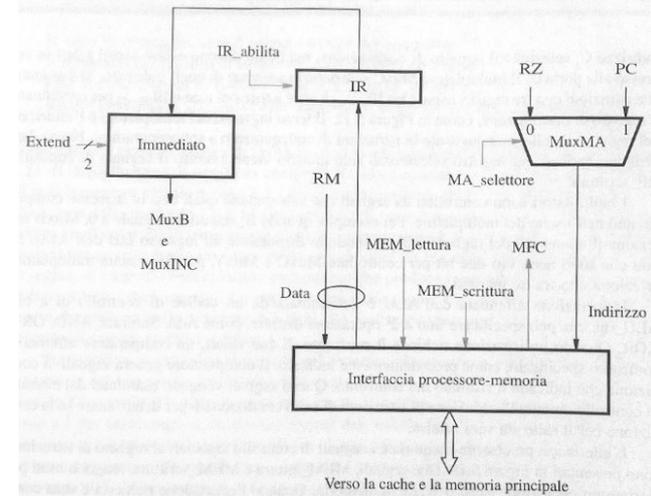
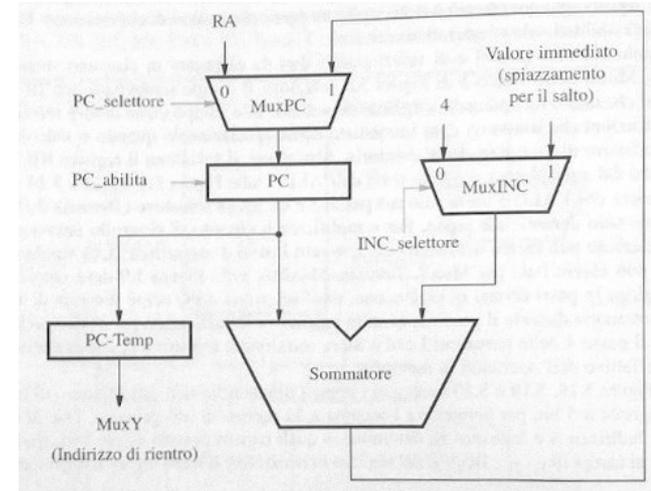
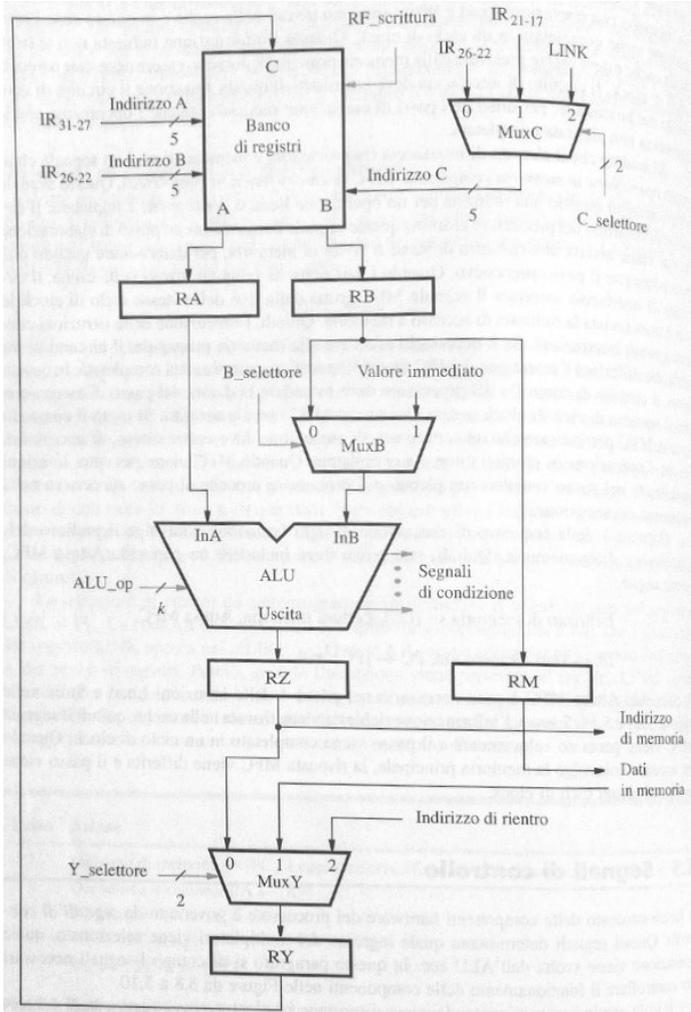
1. Indirizzo di memoria  $\leftarrow [PC]$ , Leggi memoria,  $IR \leftarrow$  Dati da memoria,  $PC \leftarrow [PC] + 4$
2. Decodifica istruzione,  $RA \leftarrow [R9]$
3.  $PC\text{-Temp} \leftarrow [PC]$ ,  $PC \leftarrow [RA]$
4.  $RY \leftarrow [PC\text{-Temp}]$
5.  $RL \leftarrow [RY]$



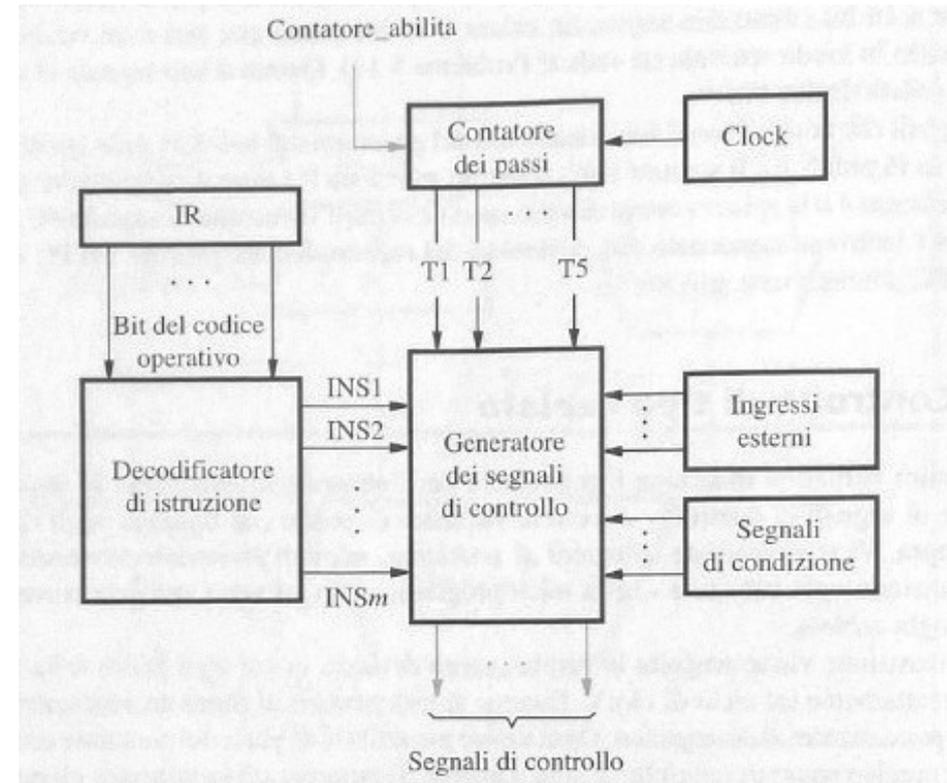
- Non sempre gli accessi alla memoria possono essere eseguiti in un ciclo di clock
- Se il dato o l'istruzione da prelevare non si trovano nella cache, l'esecuzione deve bloccarsi al passo corrente fintanto che l'operazione di memoria richiesta non è stata eseguita
- Ad operazione di memoria eseguita viene generato il segnale **MFC (memory function completed)**
- Il circuito di controllo interrompe l'esecuzione dell'istruzione finché MFC non diventa uguale a 1
- L'attesa di MFC avviene nel primo passo di ogni istruzione (prelievo istruzione dalla memoria) e nel passo 4 delle istruzioni di load e store

- Per eseguire le istruzioni macchina il processore deve generare le sequenze di segnali di controllo per ogni stadio
- I segnali di controllo consistono in:
  - Segnali di selezione per i multiplatori
  - Segnali di attivazione di alcuni registri
  - Segnali di condizione
  - Segnali per la gestione della memoria
  - Indirizzi, codice operativo e dati letti dall'istruzione nel registro IR
  - Operazione da eseguire nella ALU

# Segnali di controllo

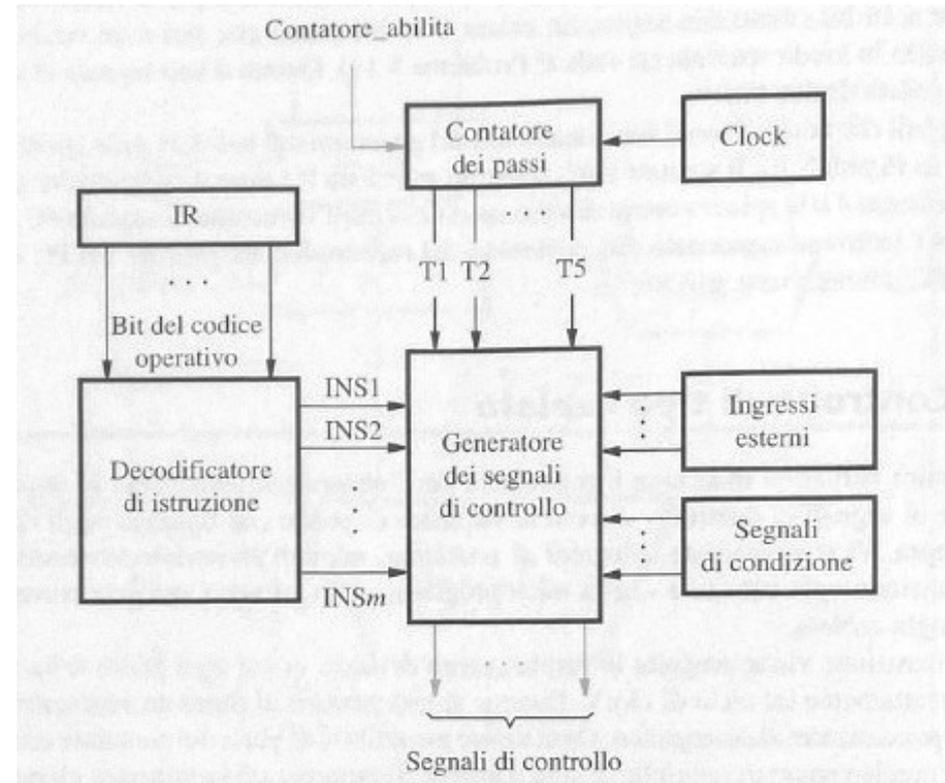


- Un approccio per generare i segnali di controllo consiste nel **CONTROLLO CABLATO**
- Esiste un **CONTATORE** modulo 5 che scandisce gli stadi di esecuzione
- Mettiamo il caso esistano  $m$  istruzioni
- Il **DECODIFICATORE DI ISTRUZIONE** genera un vettore lungo  $m$  mettendo a 1 solo il bit corrispondente all'istruzione letta su **IR**
- Il **GENERATORE DEI SEGNALE DI CONTROLLO** produce i segnali di controllo sulla base dell'istruzione in esecuzione, dello stadio attuale, dei segnali di condizione e di segnali esterni quali le interruzioni



- Gli stadi in cui si accede la memoria potrebbero durare diversi cicli di clock
- Solo negli stadi di accesso alla memoria, bisogna bloccare il contatore dei passi finché l'operazione non si è conclusa
- Il segnale WMFC è asserito quando avviene un accesso alla memoria e il segnale MFC è asserito quando si conclude
- Il segnale di abilitazione del Contatore dei passi è il seguente:

$$\text{Contatore\_abilita} = \neg\text{WMFC} + \text{MFC}$$



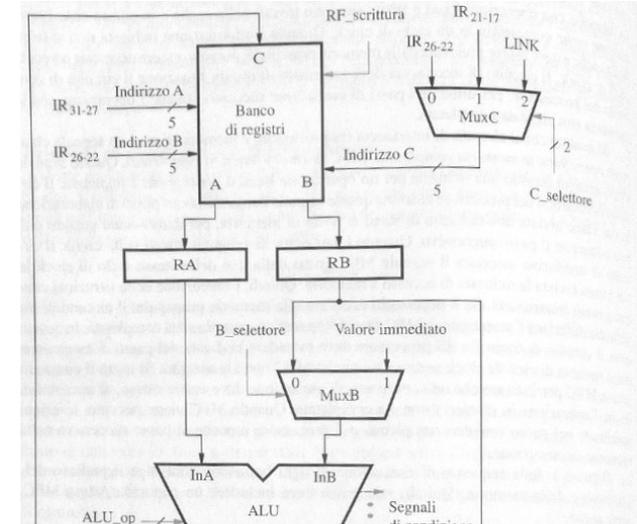
# Controllo cablato (esempi)

- Abilitazione scrittura banco registri:

$$RF\_scrittura = T5 \cdot (ALU + Load + Call)$$

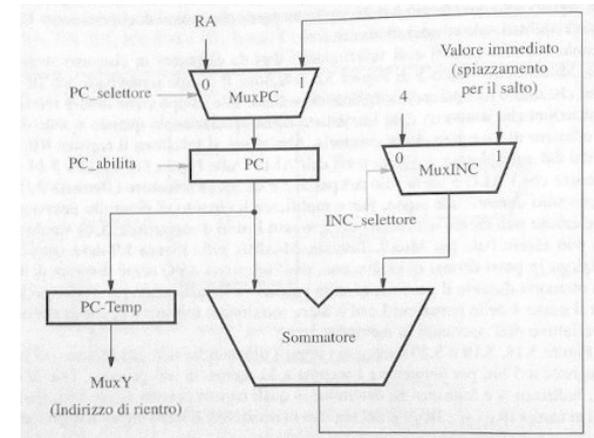
- Selettore valore immediato

$$B\_selettore = Immediato$$



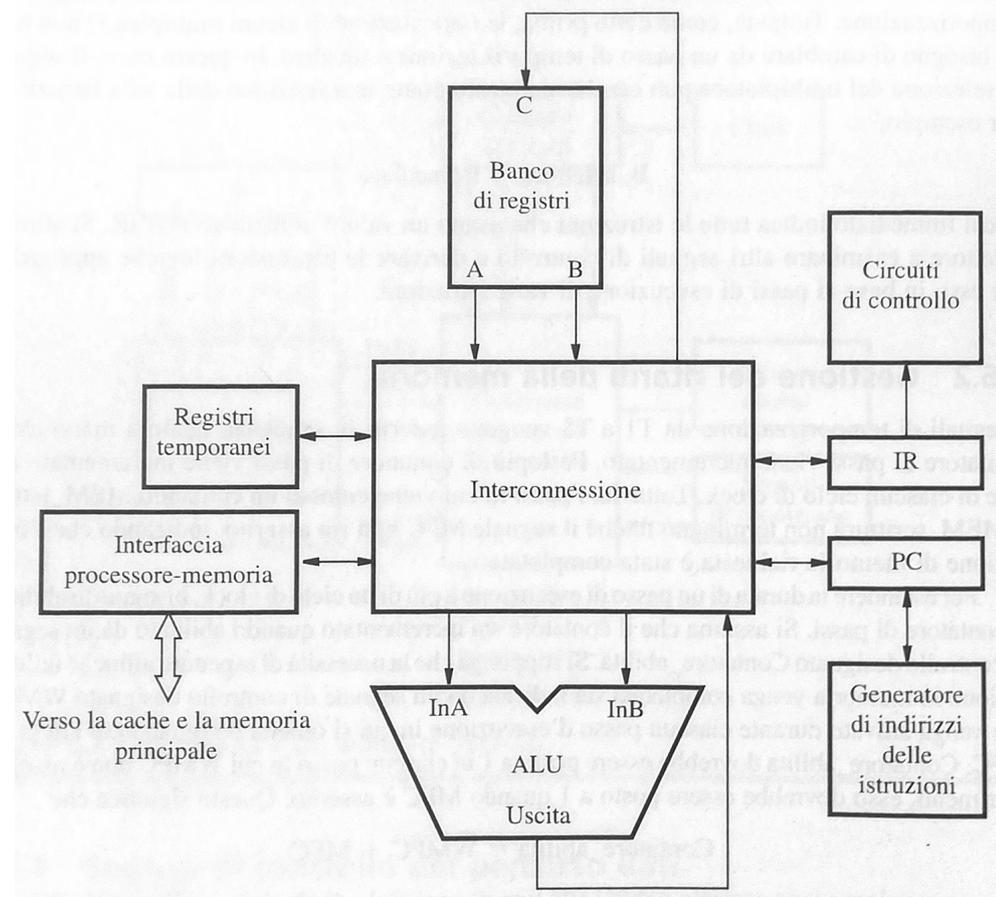
- Abilitazione registro PC

$$PC\_abilita = T1 \cdot MFC + T3 \cdot BR$$

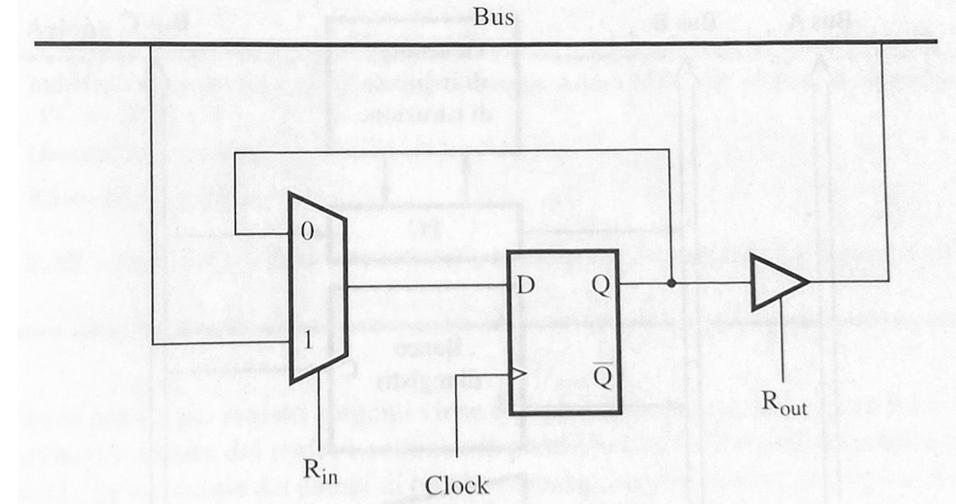


# Organizzazione processore CISC

- Le istruzioni CISC possono occupare più di una parola, non è possibile un approccio a stadi come in RISC
- Il blocco di **interconnessione** permette il trasferimento dati tra una qualsiasi coppia di componenti
- Non esistono registri interstadio, ma esistono **registri temporanei** per memorizzare risultati intermedi
- Il blocco di interconnessione è realizzato tramite **BUS**

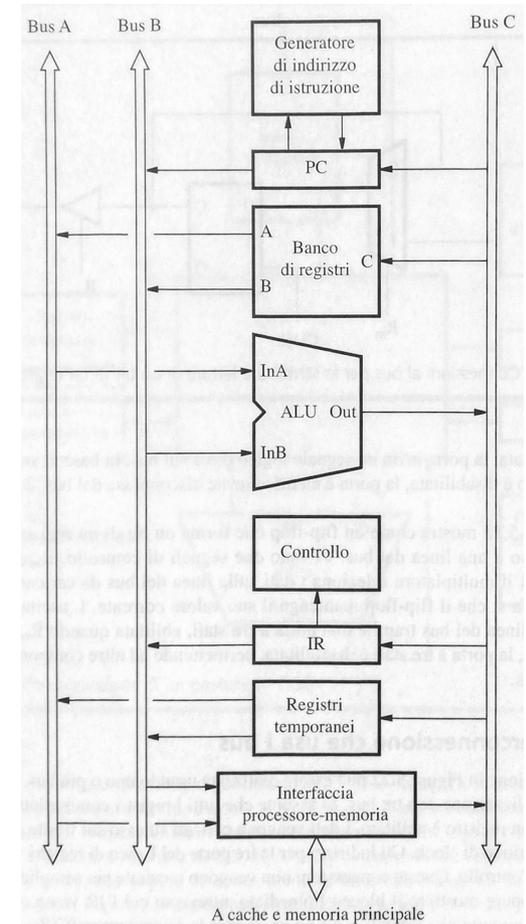


- Un **BUS** è formato da un insieme di linee a cui sono connessi vari dispositivi
- Un solo dispositivo alla volta può trasferire dati
- La porta logica che invia un segnale su una linea di BUS è chiamata **BUS DRIVER**
- I dispositivi sono collegati al BUS tramite **porte tri-state**, tutte disattivate ad eccezione del bus driver



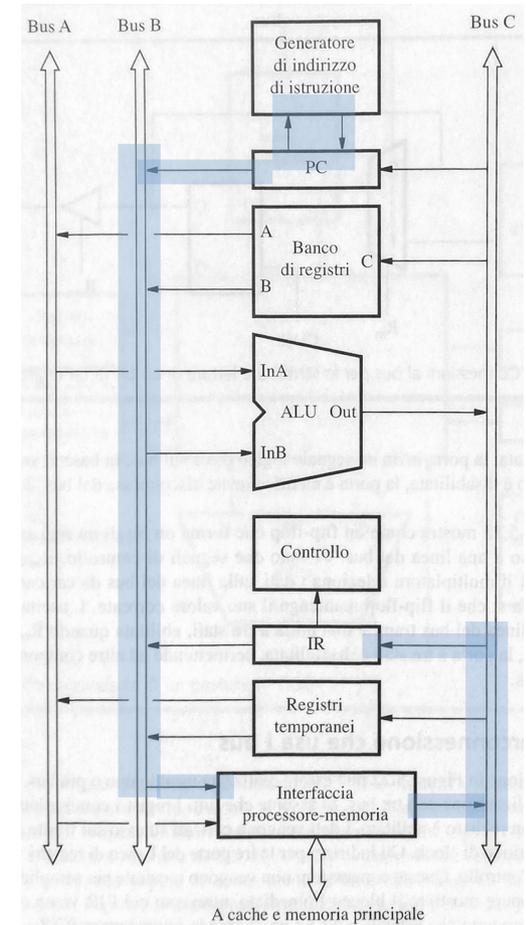
# Interconnessione con 3 BUS

- Una soluzione comune è quella di usare 3 BUS per la connessione
- Bus A e B per i sorgenti delle operazioni e Bus C per i risultati
- Tutte le componenti sono connesse ai 3 BUS
- Il generatore di indirizzo di istruzione è collegato direttamente al PC
- Il blocco di Controllo legge direttamente dall'IR
- Istruzioni diverse sono eseguite in un numero di passi differente



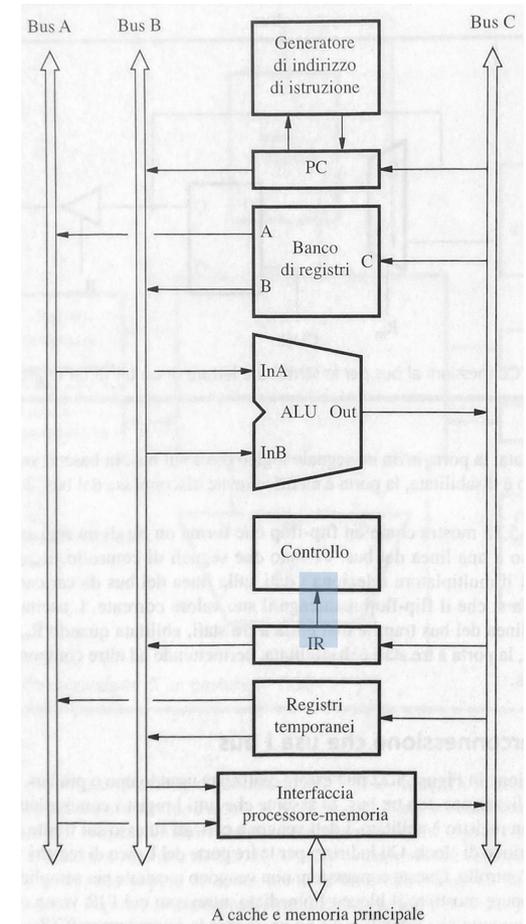
## Add R5, R6

- L'istruzione di addizione fra due registri può essere eseguita in 3 passi:
  1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4



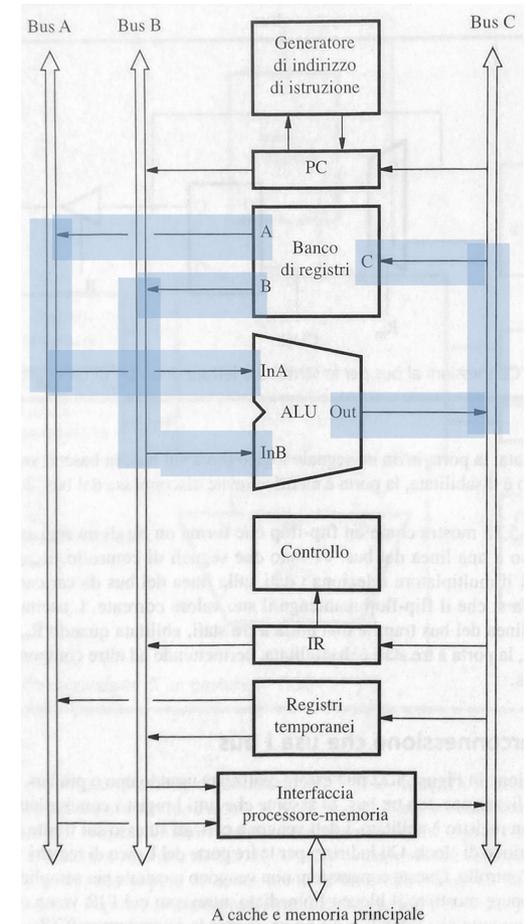
## Add R5, R6

- L'istruzione di addizione fra due registri può essere eseguita in 3 passi:
  1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  2. Decodifica Istruzione



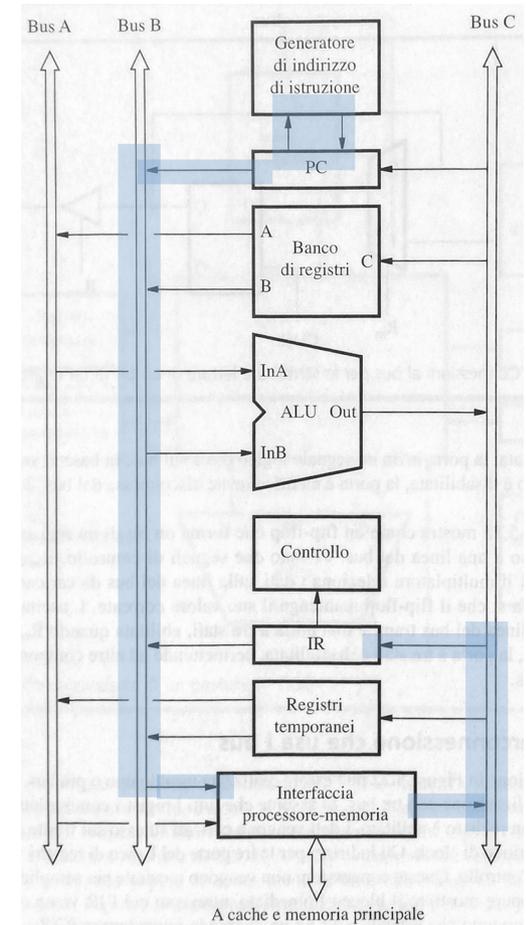
## Add R5, R6

- L'istruzione di addizione fra due registri può essere eseguita in 3 passi:
  - Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  - Decodifica Istruzione
  - R5  $\leftarrow$  [R5] + [R6]



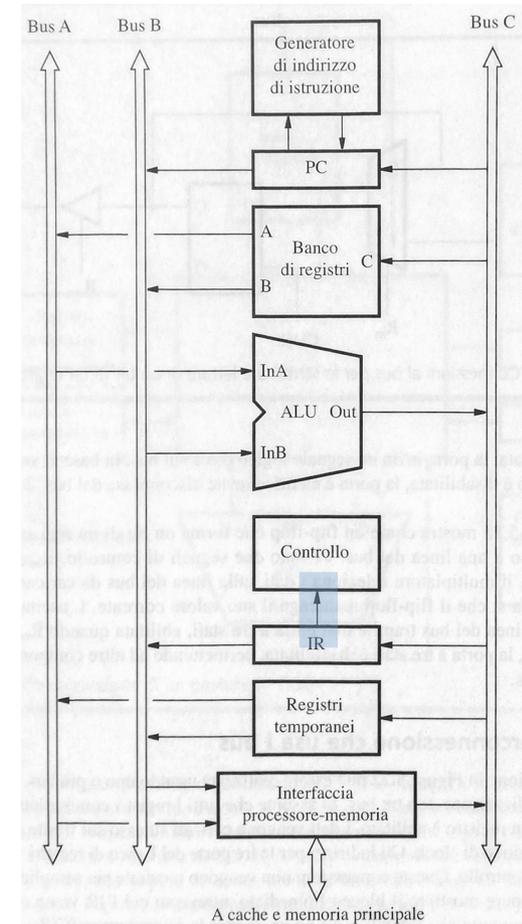
## And X(R7), R9

- L'istruzione di prodotto logico bit a bit fra una locazione di memoria (indice e spiazzamento) e un registro necessita di 7 passi:
  - Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4



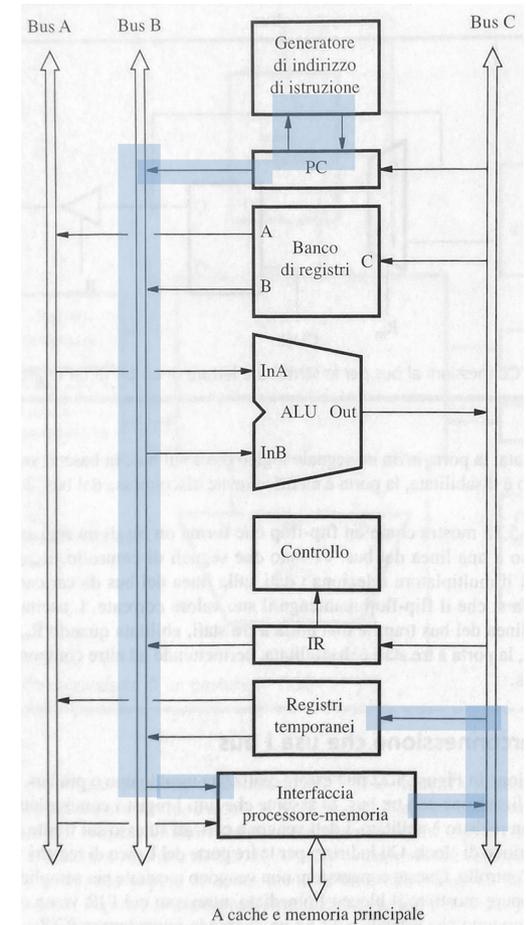
## And X(R7), R9

- L'istruzione di prodotto logico bit a bit fra una locazione di memoria (indice e spiazzamento) e un registro necessita di 7 passi:
  1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  2. Decodifica Istruzione



## And X(R7), R9

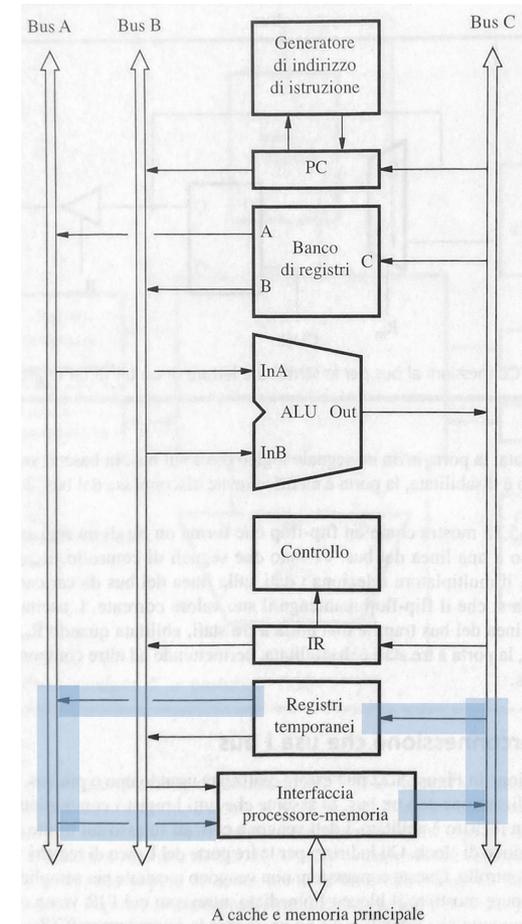
- L'istruzione di prodotto logico bit a bit fra una locazione di memoria (indice e spiazzamento) e un registro necessita di 7 passi:
  1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  2. Decodifica Istruzione
  3. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, Temp1  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4





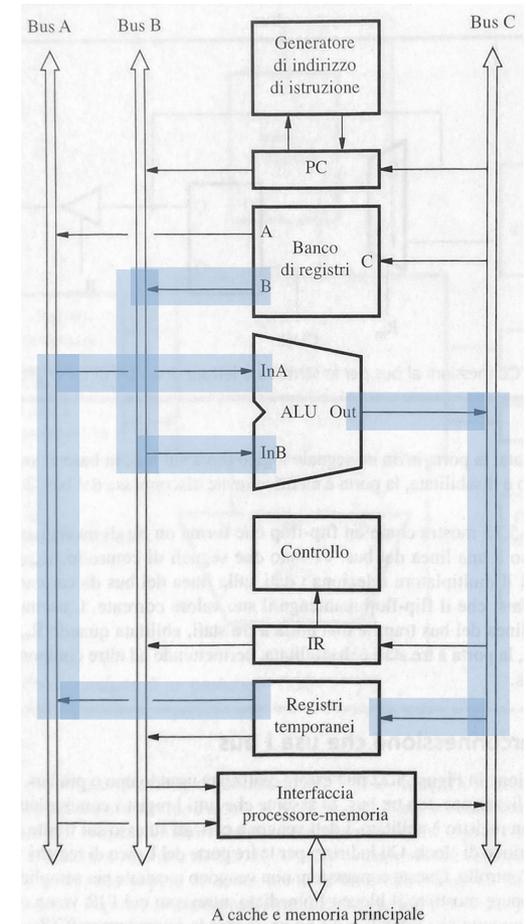
## And X(R7), R9

- L'istruzione di prodotto logico bit a bit fra una locazione di memoria (indice e spiazzamento) e un registro necessita di 7 passi:
  1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  2. Decodifica Istruzione
  3. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, Temp1  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  4. Temp2  $\leftarrow$  [Temp1] + [R7]
  5. Indirizzo di memoria  $\leftarrow$  [Temp2], Leggi memoria, Attesa MFC, Temp1  $\leftarrow$  Dati da memoria



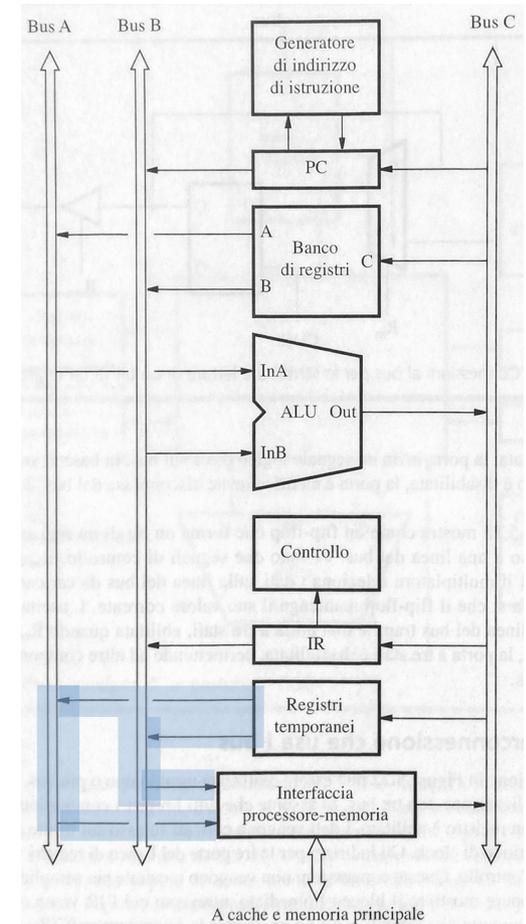
## And X(R7), R9

- L'istruzione di prodotto logico bit a bit fra una locazione di memoria (indice e spiazzamento) e un registro necessita di 7 passi:
  1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  2. Decodifica Istruzione
  3. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, Temp1  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  4. Temp2  $\leftarrow$  [Temp1] + [R7]
  5. Indirizzo di memoria  $\leftarrow$  [Temp2], Leggi memoria, Attesa MFC, Temp1  $\leftarrow$  Dati da memoria
  6. Temp1  $\leftarrow$  [Temp1] AND [R9]



## And X(R7), R9

- L'istruzione di prodotto logico bit a bit fra una locazione di memoria (indice e spiazzamento) e un registro necessita di 7 passi:
  1. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, IR  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  2. Decodifica Istruzione
  3. Indirizzo di memoria  $\leftarrow$  [PC], Leggi memoria, Attesa MFC, Temp1  $\leftarrow$  Dati da memoria, PC  $\leftarrow$  [PC] + 4
  4. Temp2  $\leftarrow$  [Temp1] + [R7]
  5. Indirizzo di memoria  $\leftarrow$  [Temp2], Leggi memoria, Attesa MFC, Temp1  $\leftarrow$  Dati da memoria
  6. Temp1  $\leftarrow$  [Temp1] AND [R9]
  7. Indirizzo di memoria  $\leftarrow$  [Temp2], Dati da memoria  $\leftarrow$  [Temp1], Scrivi in memoria, Attesa MFC



- I segnali di controllo di ogni passo vengono raccolti in una parola di memoria chiamata **microistruzione**
- L'insieme di microistruzioni rappresentanti i passi di un'istruzione macchina si chiamano **microroutine**
- Le microistruzioni di ciascuna microroutine vengono immagazzinate in locazioni consecutive della **memoria di controllo**
- Il registro **μPC** contiene l'istruzione della prossima microistruzione da caricare
- All'inizio di un'istruzione macchina il **generatore di indirizzi delle microistruzioni** carica sul μPC la prima istruzione della microroutine corrispondente
- Ogni passo μPC viene incrementato di un passo

