Architettura e funzionamento di una CPU

M. Sonza Reorda, M. Rebaudengo, L. Sterpone

Politecnico di Torino
Dipartimento di Automatica e Informatica

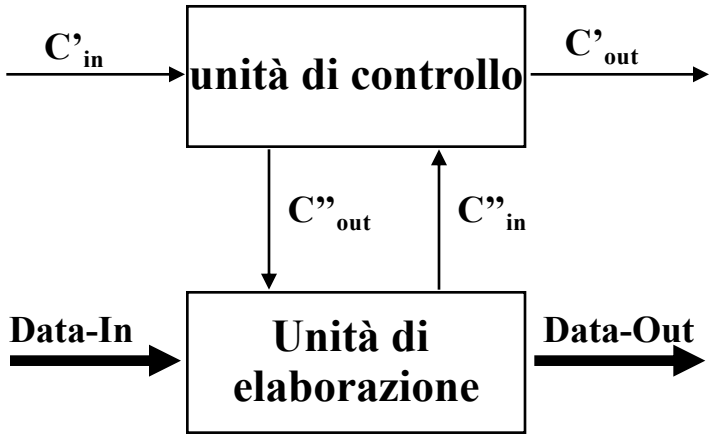
Sommario

- **Introduzione**
- **Le unità di controllo**
- **Le unità di controllo cablate**
- **Le unità di controllo microprogrammate.**

Introduzione

I processori sono composti da 2 parti:

- **l'*unità di elaborazione* (o *data-path*): contiene i registri, le unità aritmetico-logiche, ecc.**
- **l'*unità di controllo* (UC): genera i segnali di controllo per l'unità di elaborazione, sulla base dei segnali provenienti dalla stessa unità e dall'esterno.**



Funzionamento dell'unità di controllo

L'unità di controllo può essere modellata come un circuito sequenziale che riceve segnali dall'esterno (C'_{in}) e dall'unità di elaborazione (C''_{in}).

Sulla base del valore di tali segnali e dello stato corrente essa produce i segnali di controllo per l'esterno (C'_{out}) e per l'unità di elaborazione (C''_{out}).

Ad ogni colpo di clock l'UC genera i segnali di controllo necessari perché l'unità funzionale esegua una *microistruzione*.

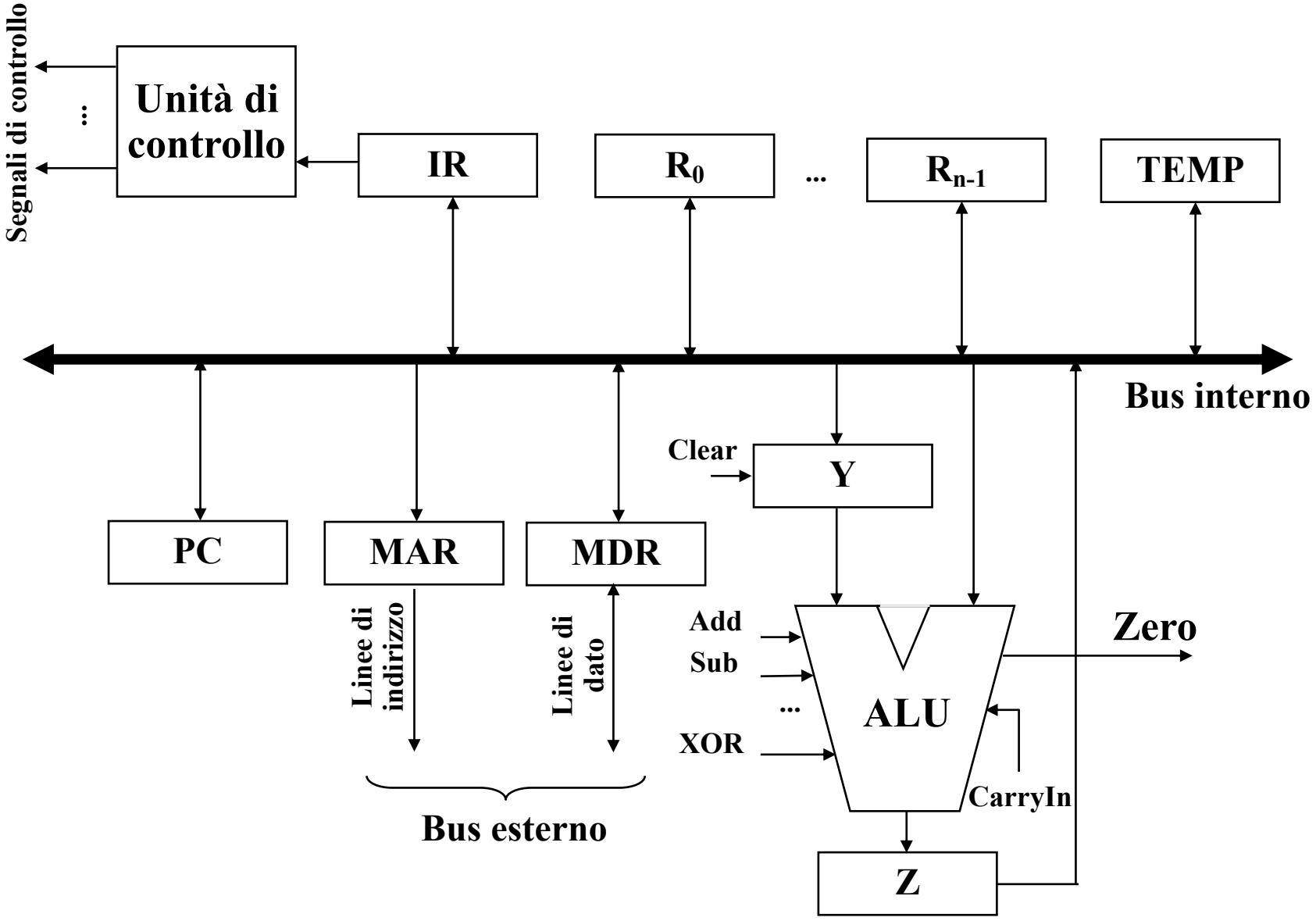
Operazioni elementari

Tutte le operazioni svolte dalla CPU possono essere ricondotte a 4 tipologie elementari:

- **prelievo di un dato o di una istruzione dalla memoria e caricamento in un registro**
- **scrittura in memoria di un dato contenuto in un registro**
- **trasferimento di un dato da un registro ad un altro**
- **esecuzione di un'operazione aritmetica o logica e memorizzazione del risultato in un registro.**

Le 4 operazioni verranno descritte con riferimento al seguente modello della CPU, semplificato rispetto a quello reale.

Architettura della CPU

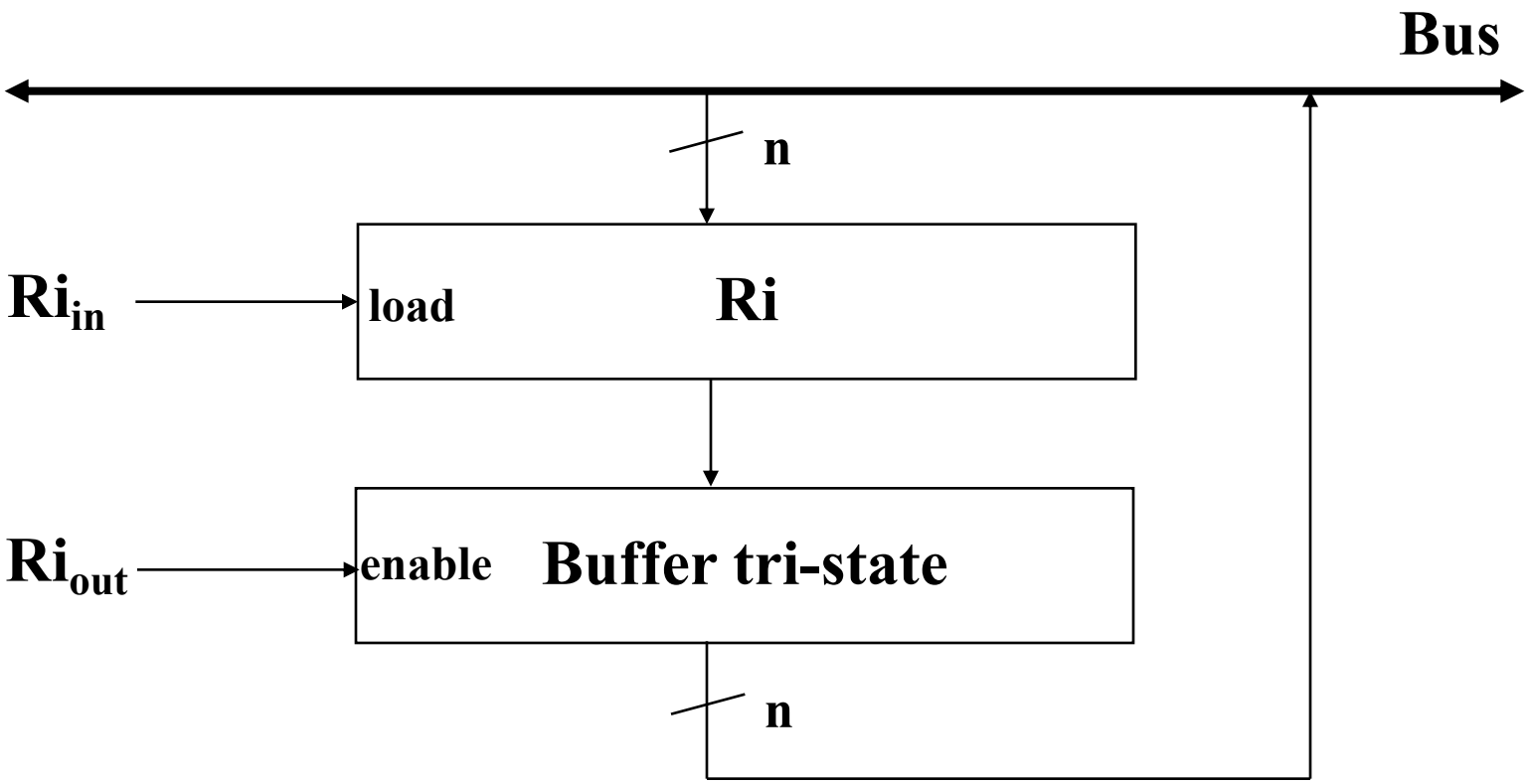


Trasferimenti tra registri

Si può supporre che ogni registro Ri connesso al bus possieda un'interfaccia controllata da due segnali:

- **Ri_{in}, che fa sì che il registro carichi il valore presente sul bus**
- **Ri_{out}, che fa sì che il contenuto del registro venga trasferito sul bus; quando il segnale non è attivo, il registro forza sul bus lo stato di *alta impedenza* (circuito aperto).**

Realizzazione dell'interfaccia



Trasferimenti tra registri

Avvengono agendo opportunamente sui segnali di interfaccia dei registri connessi al bus interno della CPU.

Ad esempio, per trasferire il contenuto del registro R1 in R4 si deve:

- **attivare $R1_{out}$: in tal modo il valore di R1 viene posto sul bus**
- **attivare $R4_{in}$: in tal modo R4 memorizza il valore presente sul bus.**

Vincoli

Per il corretto funzionamento del bus è essenziale che ad ogni istante uno e uno soltanto dei segnali Ri_{out} sia attivo.

Ad esempio non si possono eseguire nello stesso periodo di clock le due operazioni seguenti:

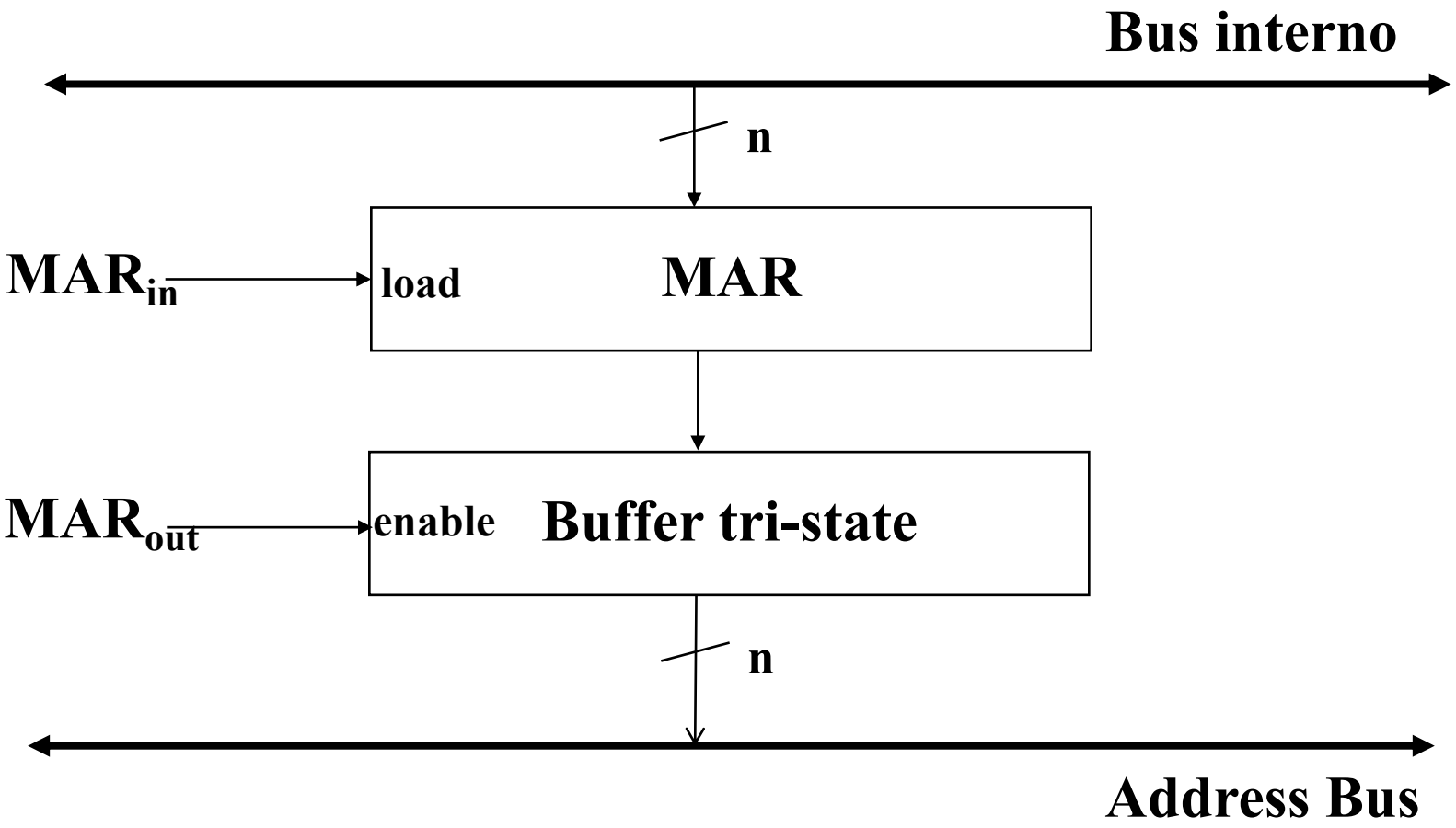
PC ← R1, R3← R2

I segnali di controllo devono essere regolati in modo che non ci siano *conflitti* attraverso operazioni di lettura e scrittura contemporanee sullo stesso registro.

Ad esempio non si possono eseguire nello stesso periodo di clock le due operazioni seguenti :

MAR ← R1, R1← R2

Connessione di MAR



Accesso all'address bus

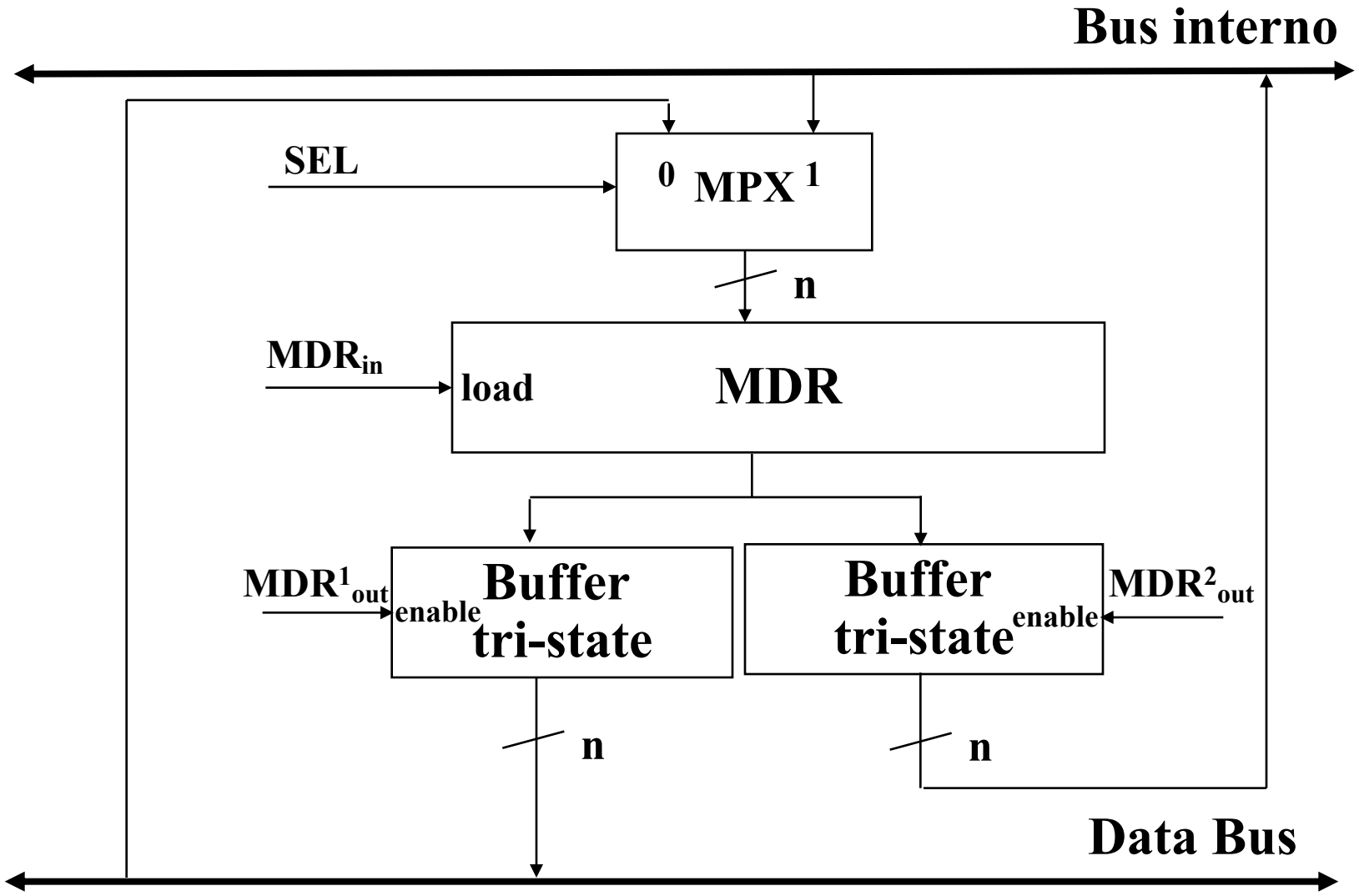
Per accedere all'address bus (per un'operazione di accesso in memoria) occorre effettuare la seguente microistruzione:

MAR ← R1

che richiede l'attivazione dei seguenti segnali di controllo:

- **attivare $R1_{out}$: in tal modo il valore di R1 viene posto sul bus interno**
- **attivare MAR_{in}: in tal modo MAR memorizza il valore presente sul bus**
- **attivare MAR_{out}: in tal modo il valore di MAR viene posto sull'address bus.**

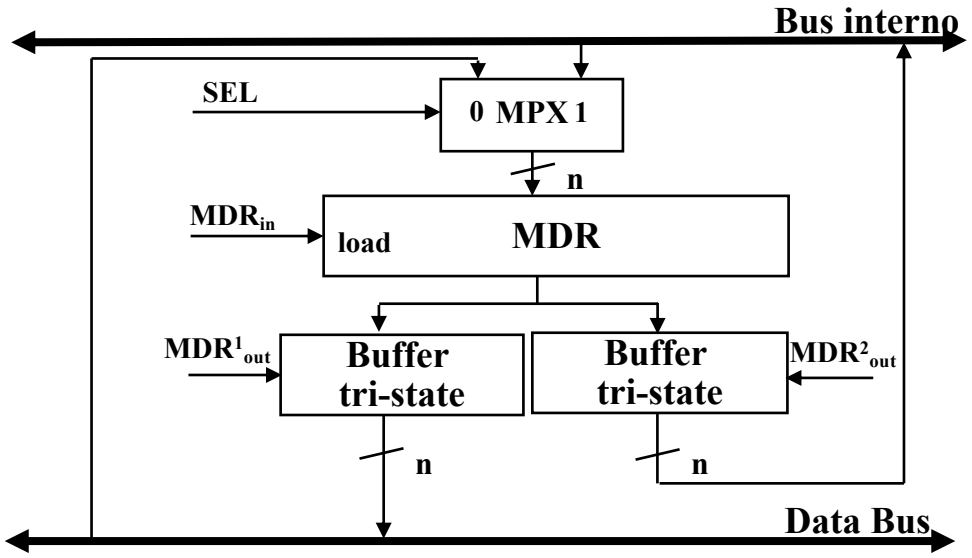
Connessione di MDR



Connessione di MDR

MDR gestisce due operazioni:

- **Scrittura in memoria**
- **Lettura dalla memoria.**



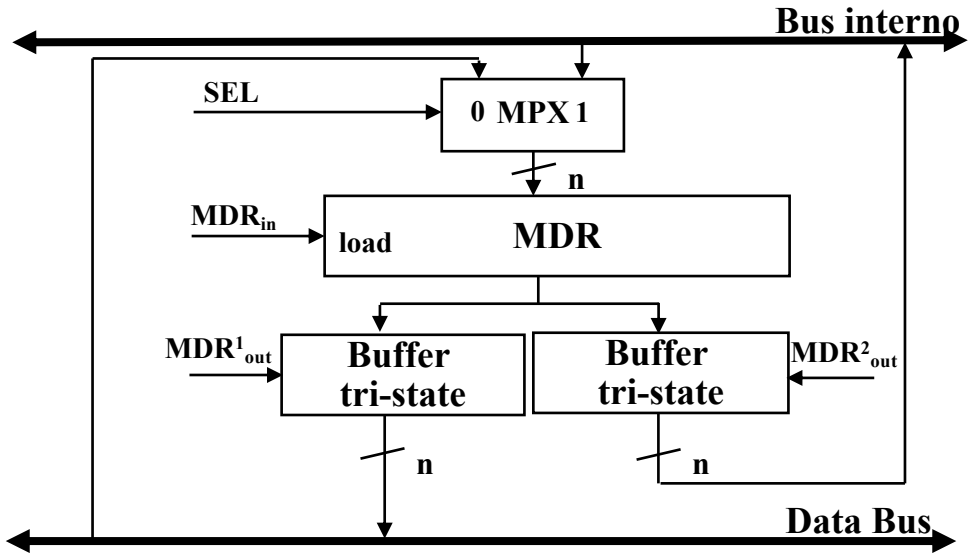
Connessione di MDR

Scrittura in memoria:

- **MDR ← Bus interno**
- **Accesso in memoria**

Segnali di controllo abilitati:

- **SEL=1**
- **MDR_{in}**
- **MDR^1_{out}**
- **Accesso in memoria.**



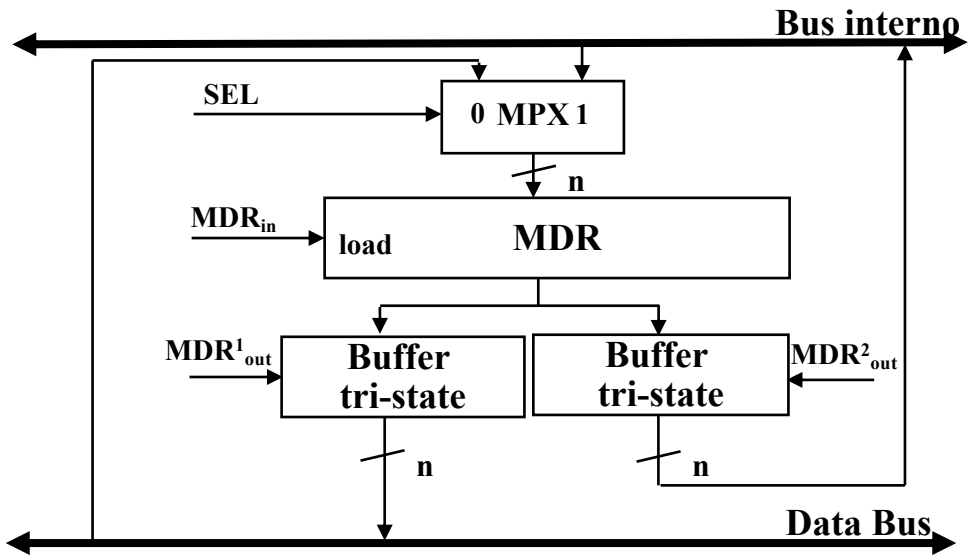
Connessione di MDR

Lettura dalla memoria:

- **MDR ← Bus esterno**
- **R ← MDR**

Segnali di controllo abilitati:

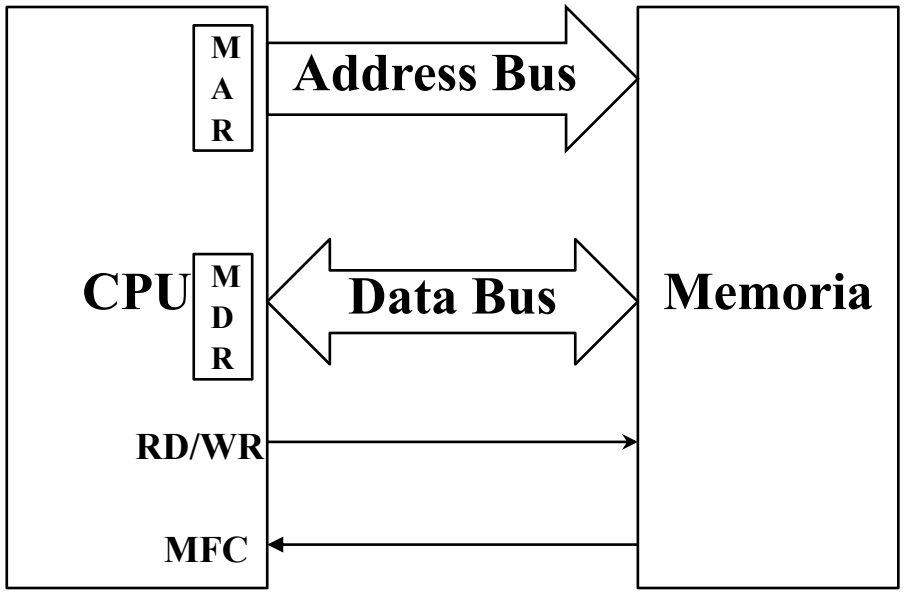
- **SEL=0**
- **MDR_{in}**
- **MDR^2_{out}**
- **R_{in}**



Temporizzazioni

Ciascun segnale deve rimanere attivo per il tempo necessario a completare l'operazione da esso pilotata, tenuto conto dei ritardi dei vari componenti attivati.

Interfaccia CPU - Memoria



La memoria

- **dopo aver ricevuto i segnali di lettura/scrittura impiega un certo tempo a completare l'operazione (acquisire il dato in scrittura o mettere sul bus il dato in lettura)**
- **una volta completata l'operazione, attiva il segnale MFC (*Memory Function Completed*).**

Tale meccanismo presuppone un bus di tipo asincrono.

Prelievo di una parola dalla memoria

Si supponga che R1 contenga l'indirizzo della locazione di memoria che deve essere caricata in R2 (*LOAD R2, [R1]*).

- **MAR ← R1**
- **Attiva il segnale di lettura**
- **Aspetta sino al segnale MFC**
- **MDR ← Bus esterno**
- **R2 ← MDR**

Prelievo di una parola dalla memoria

- **MAR ← R1**
- **Attiva il segnale di lettura**
- **Aspetta sino al segnale MFC**
- **MDR ← Bus esterno**
- **R2 ← MDR**

Segnali di controllo attivati (nell'ordine)

- **$R1_{out}$**
- **MAR_{in}**
- **MAR_{out}**
- **RD**
- **Aspetta MFC**
- **SEL = 0**
- **MDR_{in}**
- **MDR^2_{out}**
- **$R2_{in}$**
- **End**

Scrittura di una parola in memoria

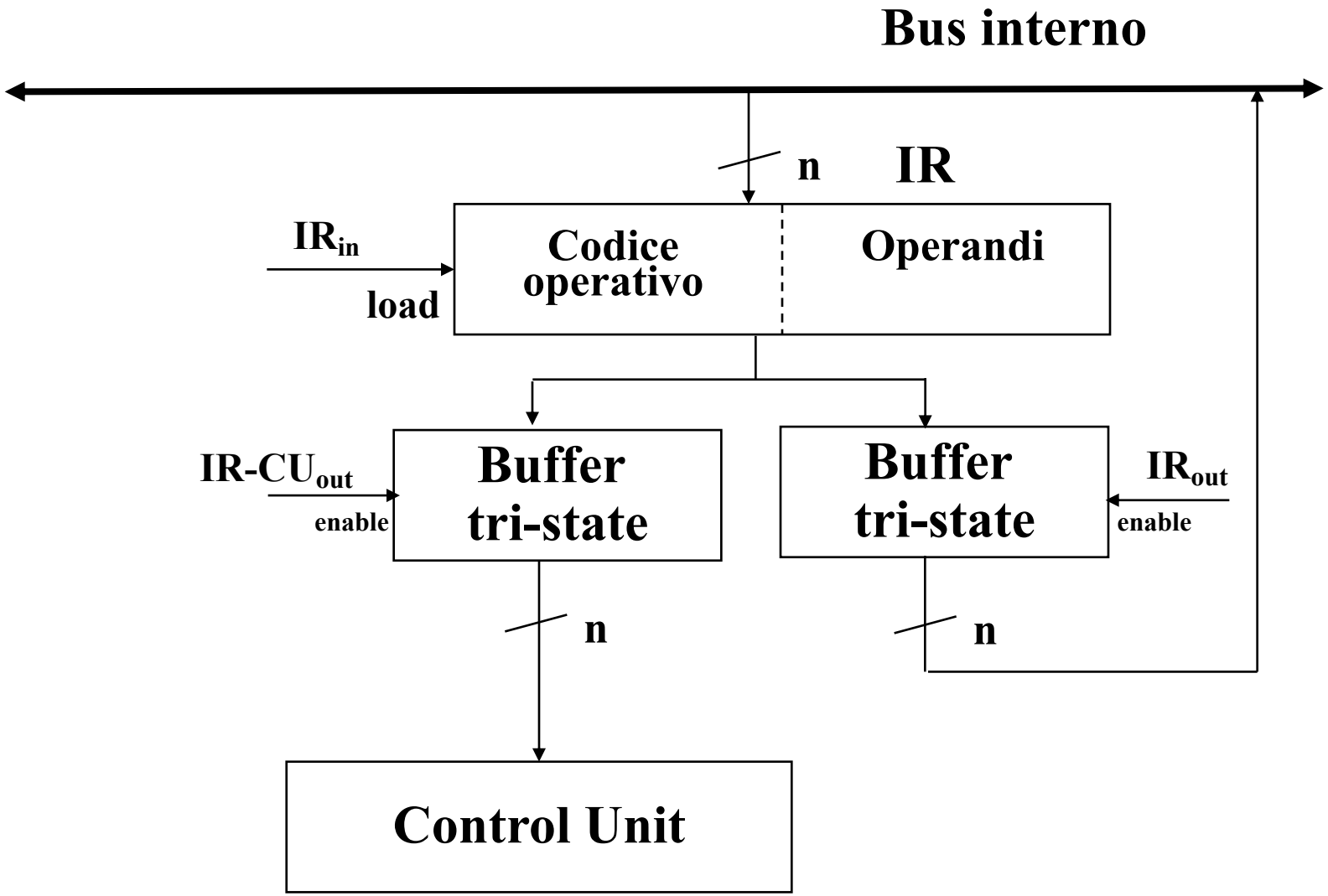
Si supponga che R1 contenga il dato che deve essere caricato nell'indirizzo di memoria contenuto in R2 (*STORE R1, [(R2]*)

- **MDR ← R1**
- **MAR ← R2**
- **Attiva il segnale di scrittura**
- **Aspetta sino al segnale MFC**

Segnali di Controllo abilitati:

- **$R1_{out,}$**
- **SEL = 1,**
- **MDR_{in}**
- **MDR^1_{out}**
- **$R2_{out}$,**
- **MAR_{in}**
- **MAR_{out},**
- **WR**
- **Aspetta MFC**
- **End**

Connessione di IR



Fetch

Durante la fase di fetch si combinano due operazioni contemporaneamente:

- **Lettura della prossima istruzione da eseguire dalla memoria**
- **Aggiornamento del Program Counter.**

Fetch

Microistruzioni

- **MAR ← PC**
- **Y = 0**
- **Carry = 1**
- **Z ← PC + Y + Carry**
- **Attiva il segnale di lettura**
- **PC ← Z**
- **Aspetta sino al segnale MFC**
- **MDR ← Bus esterno**
- **IR ← MDR**

Segnali di controllo attivati (nell'ordine)

- **PC_{out}**
- **MAR_{in}**
- **Clear Y**
- **Set Carry In to ALU**
- **Add**
- **Z_{in}**
- **MAR_{out}**
- **RD**
- **Aspetta MFC**
- **Z_{out}**
- **PC_{in}**
- **SEL = 0**
- **MDR_{in}**
- **MDR^{2}_{out}**
- **IR_{in}**

Fetch

Il numero di microistruzioni (e quindi periodi di clock) necessari per eseguire un'operazione può essere ridotto attivando più segnali di controllo nella stessa microistruzione, ove possibile.

Segnali di controllo attivati (nell'ordine)

- PC_{out}
- MAR_{in}
- Clear Y, MAR_{out}
- Set Carry In to ALU, RD
- Add
- Z_{in}
- Z_{out}
- Pc_{in}
- Aspetta MFC
- SEL = 0
- MDR_{in}
- MDR^2_{out}
- IR_{in}

Fetch

Il numero di microistruzioni (e quindi periodi di clock) necessari per eseguire un'operazione può essere ridotto attivando più segnali di controllo nella stessa microistruzione, ove possibile.

Segnali di controllo attivati (nell'ordine)

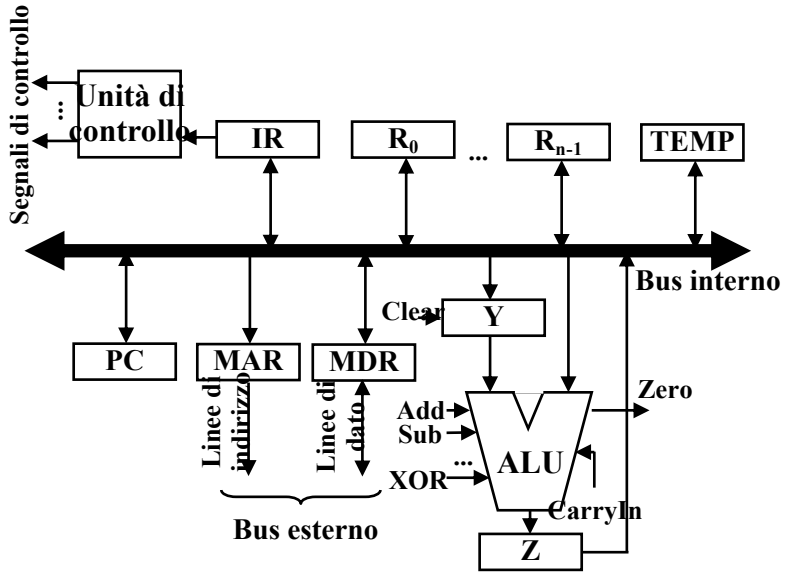
- PC_{out}
- MAR_{in}
- Clear Y, MAR_{out}
- Set Carry In to ALU, RD
- Add
- Z_{in}
- Z_{out}
- Pc_{in}
- Aspetta MFC
- SEL = 0
- MDR_{in}
- MDR^2_{out}
- IR_{in}

Queste 4 microistruzioni vanno eseguite solo dopo che MFC è stato attivato e possono quindi andare parzialmente in parallelo alle precedenti

Esecuzione di un'operazione logica o aritmetica

Per eseguire la somma tra i registri R1 ed R2 e mettere il risultato in R3 (*ADD R3, R1, R2*) si devono eseguire le seguenti microistruzioni:

- **Y ← R1**
- **Z ← Y + R2**
- **R3 ← Z**



I segnali di controllo da attivare sono (nell'ordine):

- **$R1_{out}$**
- **Y_{in}**
- **$R2_{out}$**
- **Add**
- **Z_{in}**
- **Z_{out}**
- **$R3_{in}$**
- **End**

Esempio

Si consideri un'istruzione che scrive in R1 il contenuto della cella di memoria il cui indirizzo è dato dalla somma di R2 e di una costante 23 (*LOAD R1, 23 (R2)*).

Ignorando la fase di Fetch, le microistruzioni necessarie sono:

- **Y ← R2**
- **Bus int ← (Campo operando di IR)**
- **Z ← Add**
- **MAR ← Z**
- **Attiva il segnale di lettura**
- **Aspetta sino al segnale MFC**
- **MDR ← Bus esterno**
- **R1 ← MDR**

I segnali di controllo sono i seguenti:

- **$R2_{out}$**
- **Y_{in}**
- **$(Campo\ operando\ di\ IR)_{out}$**
- **Add**
- **Z_{in}**
- **Z_{out}**
- **MAR_{in}**
- **RD**
- **Aspetta MFC**
- **SEL = 0**
- **MDR_{in}**
- **MDR^2_{out}**
- **$R1_{in}$**
- **End**

Istruzioni di salto incondizionato

La microistruzione che deve essere eseguita in corrispondenza di una istruzione di salto incondizionato è (a parte la fase di Fetch):

- **PC ← (Campo operando di IR)$_{out}$**

I segnali che devono essere attivati sono:

- **(Campo operando di IR)$_{out}$**
- **PC$_{in}$**

Si suppone che l'operando immediato corrisponda all'indirizzo dell'istruzione a cui saltare.

Salto condizionato

Consideriamo l'istruzione BNE R1, R2, lab

Le microistruzioni da eseguire sono le seguenti:

- **Y ← R1**
- **Z ← Y - R2**
- **Y ← PC, if Zero = 0 then**
    - **Z ← (Campo operando di IR)$_{out}$ + Y**
    - **PC ← Z**

Nota:

- **Zero = 1 se Z = 0**
- **Zero = 0 se Z ≠ 0**

I segnali che devono essere attivati sono (nell'ordine):

- **R1$_{out}$**
- **Y$_{in}$**
- **R2$_{out}$**
- **Sub**
- **Z$_{in}$**
- **PC$_{out}$**
- **Y$_{in}$**
- **IF Zero = 0 then**
    - **(Campo operando di IR)$_{out}$**
    - **Add**
    - **Z$_{in}$**
    - **Z$_{out}$**
    - **PC$_{in}$**
- **End**

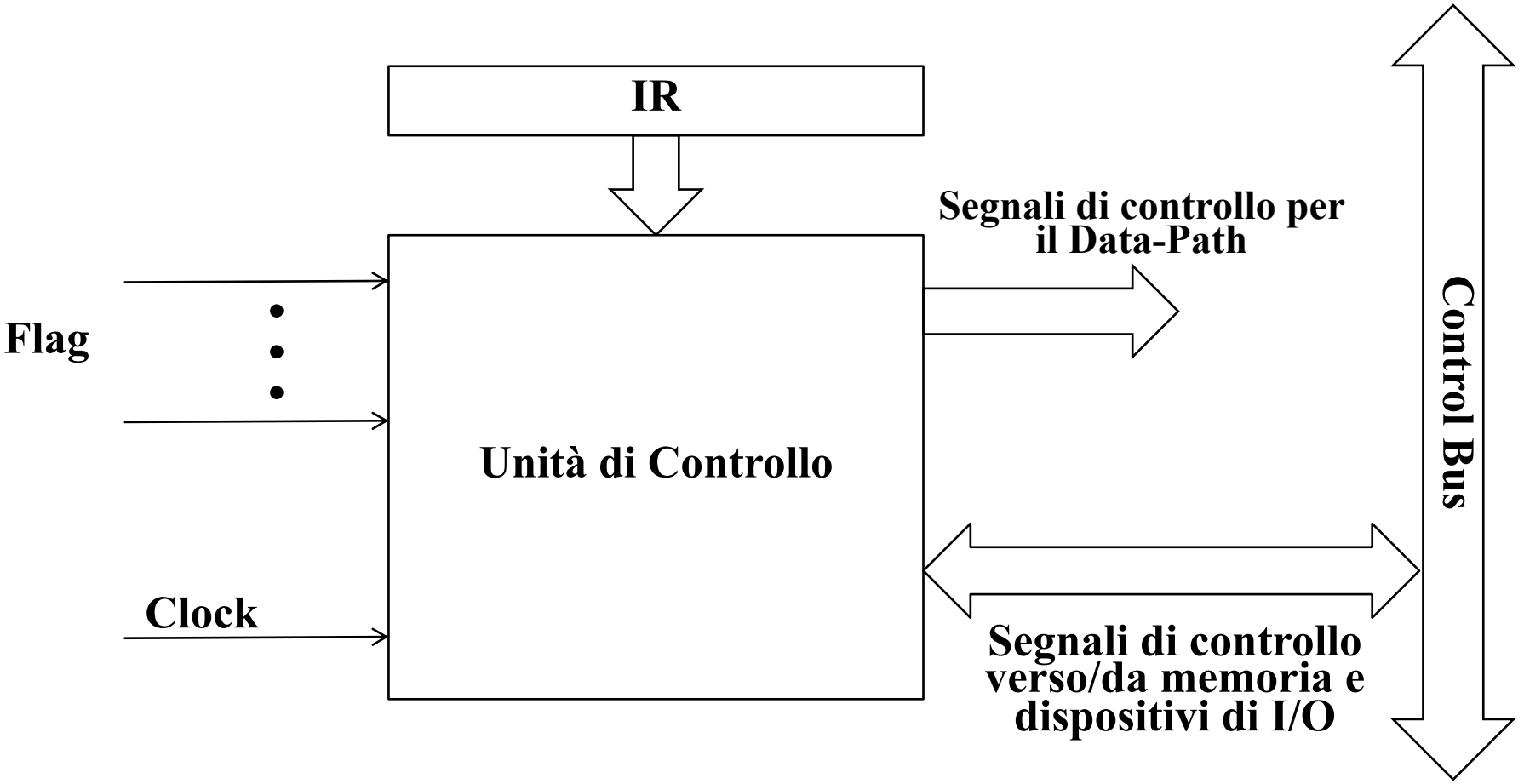
Flag di condizione

In alcuni processori (ad esempio quelli della famiglia x86) il meccanismo di gestione dei salti condizionati è diverso.

In tali processori

- **Esistono alcuni bit (denominati *flag di condizione*) che sono automaticamente forzati a 0 o 1 dalle istruzioni aritmetiche sulla base del risultato prodotto**
  - **ZF (*Zero Flag*): forzato a 1 se il risultato è nullo, a 0 diversamente**
  - **SF (*Sign Flag*): forzato a 1 se il risultato è negativo, a 0 diversamente**
- **Esistono istruzioni che eseguono il salto se una certa condizione su uno o più flag di condizione è verificata.**
  Esempio
  JZ lab salta se Z=1

Progetto dell'Unità di Controllo



Progetto dell'unità di controllo

Si parte generalmente da una descrizione del suo funzionamento, basata ad es. su un diagramma a stati.

Si esegue poi la trasformazione in hardware.

Esistono 2 strategie:

- **unità di controllo *cablate* (o *hardwired*): la UC viene considerata come un normale circuito sequenziale, a cui applicare i metodi tradizionali di progetto**
- **unità di controllo *microprogrammate*: ogni operazione che l'unità di controllo deve eseguire viene descritta da una microistruzione; i valori da assegnare ai segnali di controllo in corrispondenza di ogni microistruzione sono immagazzinati in un'apposita memoria.**

Obiettivi

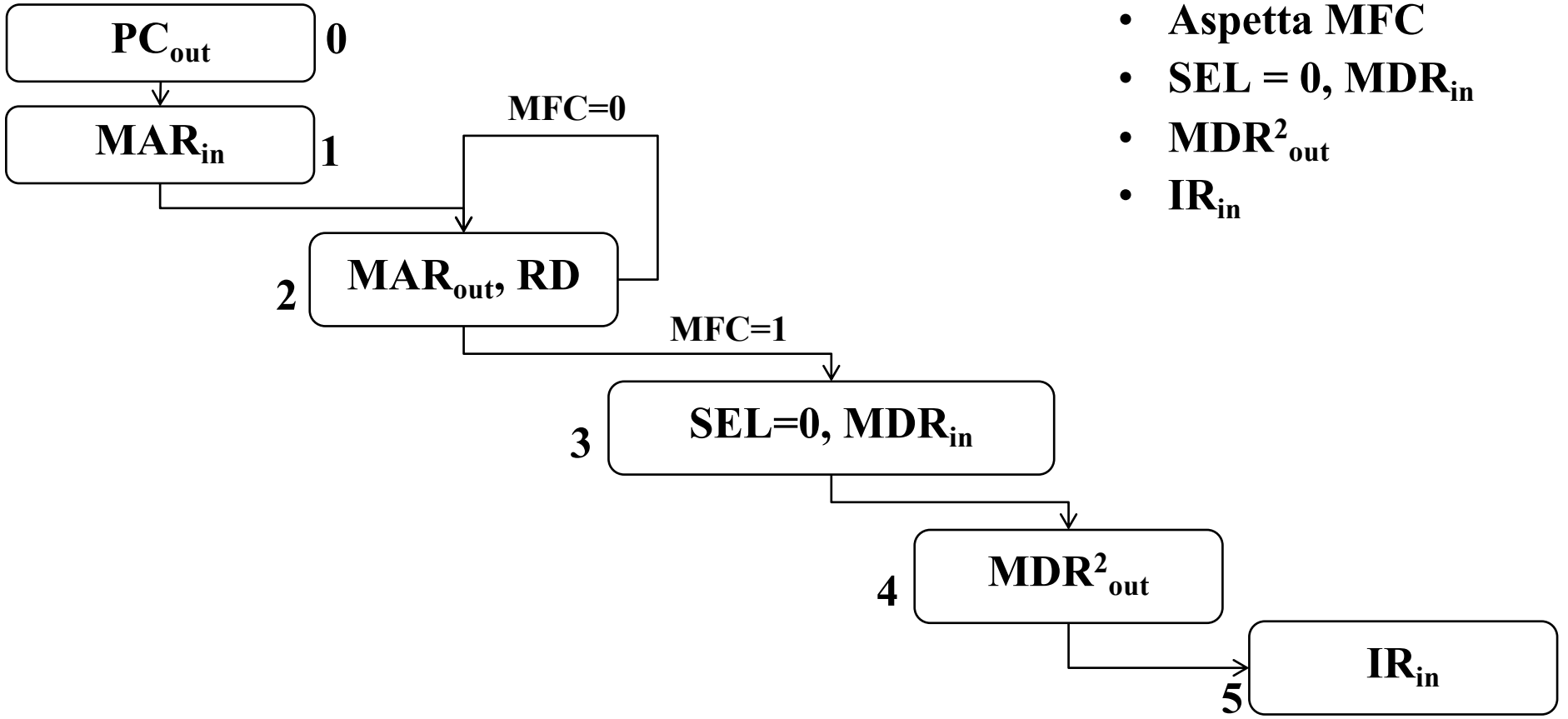
- **Minimizzare la quantità di hardware**
- **Massimizzare la velocità di esecuzione**
- **Ridurre il tempo di progetto (flessibilità).**

Progetto di una UC cablata

L’UC è modellata come una Macchina a Stati Finiti (*Finite State Machine* o FSM).

Esempio

La parte di grafo degli stati dell'UC che descrive la fase di fetch (senza l'aggiornamento del PC) è la seguente:



- PC_{out}
- MAR_{in}
- MAR_{out}, RD
- Aspetta MFC
- SEL = 0, MDR_{in}
- MDR^2_{out}
- IR_{in}

Esempio

La corrispondente tabella degli stati è:

Stato corrente	input	segnali di controllo	stato futuro
0	- - - - - - - - - - - - - - - -	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1
1	- - - - - - - - - - - - - - - -	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	2
2	0 - - - - - - - - - - - - - - -	0 0 1 0 0 0 0 0 0 1 0 0 0 0	2
2	1 - - - - - - - - - - - - - - -	0 0 1 0 0 0 0 0 0 1 0 0 0 0	3
3	- - - - - - - - - - - - - - - -	0 0 0 1 0 0 0 0 0 0 0 0 1 0	4
4	- - - - - - - - - - - - - - - -	0 0 0 0 1 0 0 0 0 0 0 0 0 0	5
5	- - - - - - - - - - - - - - - -	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	6

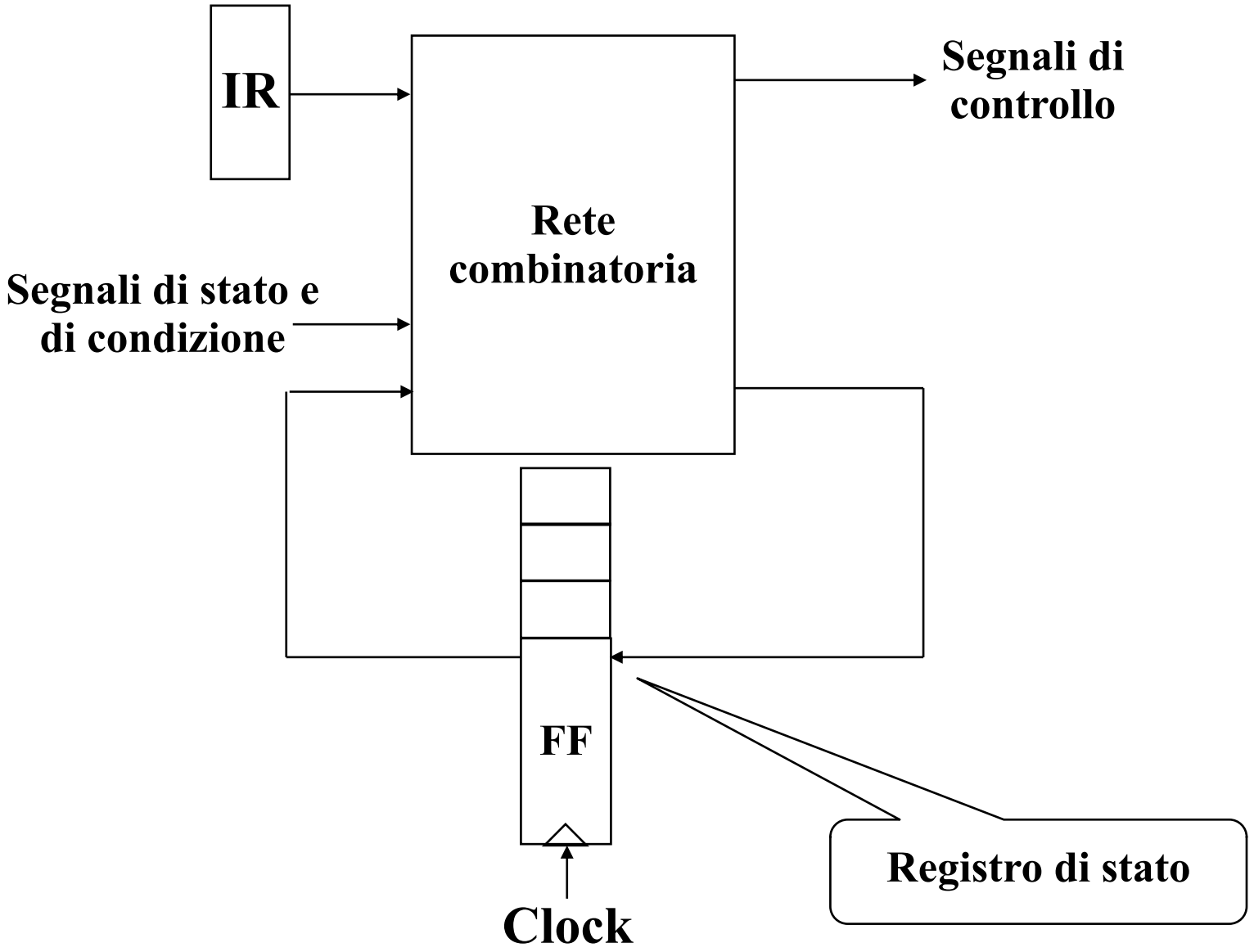
Progetto e Sintesi

Lo stato dell'FSM è caratterizzato ad ogni colpo di clock dal valore presente in un apposito registro di stato.

Per ogni stato, in corrispondenza di ciascuna possibile combinazione di ingressi (provenienti dall'esterno e dall'unità di elaborazione), si definiscono le combinazioni di uscita (segnali di controllo verso l'unità di elaborazione) e lo stato futuro.

Si costruisce così una tabella degli stati, che viene poi trattata con i normali metodi di sintesi, e da essa si genera l'hardware corrispondente.

Architettura



Complessità dell'unità di controllo

La complessità dell'unità di controllo è in generale proporzionale al prodotto tra

- **il numero di segnali di controllo (provenienti dall'esterno e dall'unità di elaborazione)**
- **il numero di stati.**

UC cablate: vantaggi

- **Minimizza l'area di silicio utilizzata**
- **Garantisce la massima frequenza di funzionamento (dell'UC)**
- **Permette di minimizzare il costo del circuito**

UC cablate: svantaggi

- **Le dimensioni della macchina a stati finiti possono essere tali da renderla difficile da progettare**
- **Una volta che il circuito è stato sintetizzato (ossia la macchina a stati finiti è stata trasformata nell'hardware corrispondente), questo non ha una struttura regolare, e la sua modifica è in genere estremamente complessa.**

Le unità di controllo microprogrammate

- **Introduzione**
- **Caratteristiche**
- **Microprogrammazione verticale**

Introduzione

La microprogrammazione fu proposta per la prima volta da M.V. Wilkes nel 1951 (articolo "The Best Way to Design an Automatic Calculating Machine").

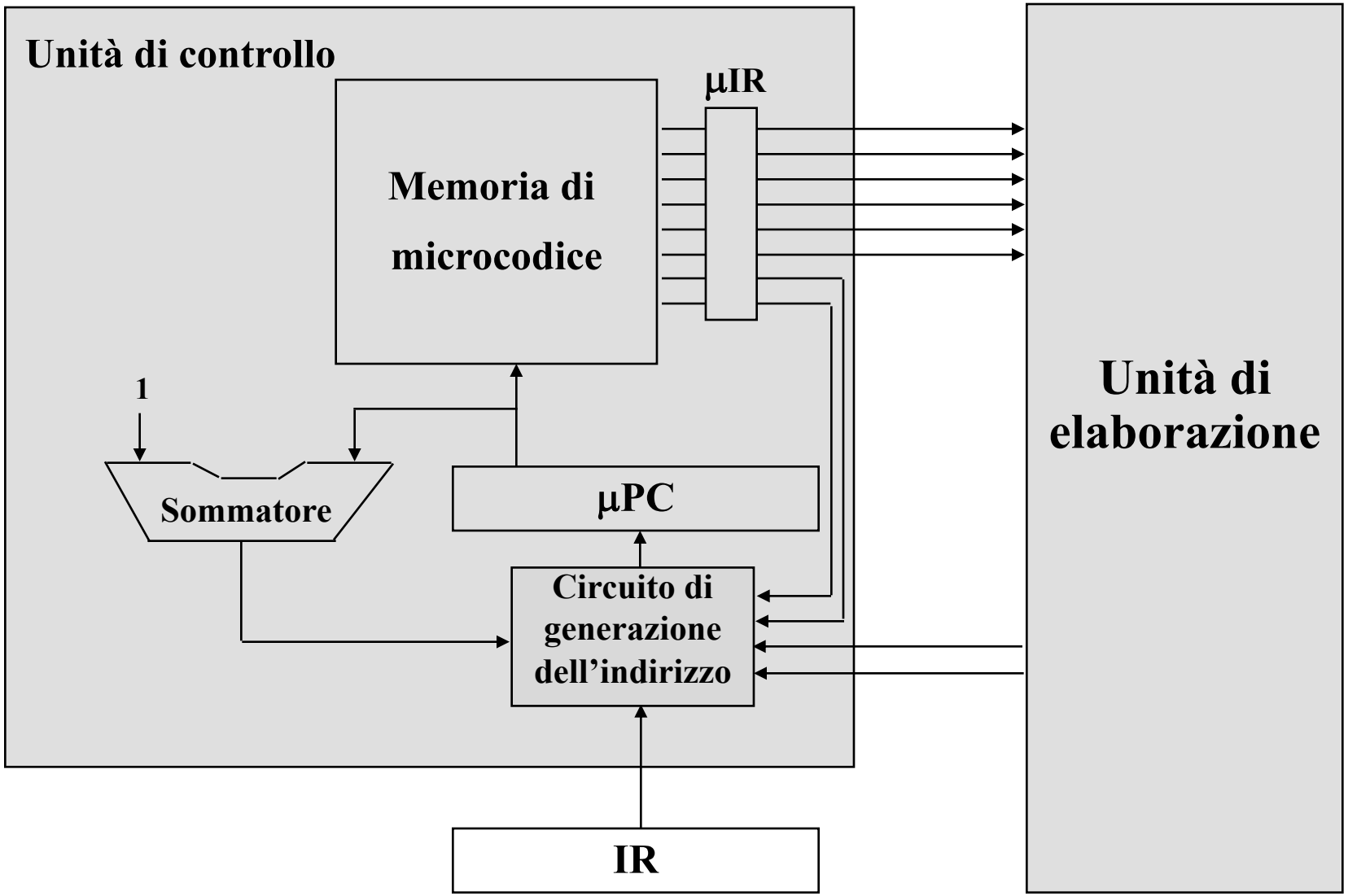
L'insieme di segnali di controllo prodotti dalla UC ad un certo istante viene denominato *parola di controllo* (Control Word, CW), o *microistruzione*.

Ciascuna istruzione corrisponde ad una sequenza di microistruzioni.

L'insieme delle microistruzioni (*microprogramma*) è memorizzato in una memoria apposita (*memoria di microcodice*) in un formato prefissato.

Moltissimi processori CISC includono un'UC microprogrammata.

Architettura



Funzionamento

- **Si esegue una lettura dalla *Memoria di Microcodice*, utilizzando il contenuto del *Micro Program Counter* (μPC) come indirizzo**
- **La parola corrispondente viene caricata nel *Micro Instruction Register* (μIR)**
- **Il contenuto del μIR pilota i segnali di controllo per l'unità di elaborazione e per la logica di generazione dell'indirizzo della successiva microistruzione**
- **Tale logica genera un nuovo indirizzo, sulla base anche dei segnali provenienti dall'esterno (ad esempio dall'Instruction Register).**

Tutte queste operazioni vengono eseguite in un solo colpo di clock.

Generazione dell'indirizzo della microistruzione successiva

L'indirizzo della microistruzione successiva può essere (a seconda delle microistruzioni):

- **quello successivo**
- **un indirizzo fornito dall'esterno (ad esempio l'inizio del microcodice dell'esecuzione di una nuova istruzione)**
- **un indirizzo di salto (condizionato o incondizionato).**

Formato delle microistruzioni

Dipende da come si risolvono 2 problemi:

- **il calcolo dell'indirizzo della successiva microistruzione**
- **la codifica dei segnali di controllo.**

Indirizzo della microistruzione successiva

Se non si hanno microistruzioni di salto è dato da quello della microistruzione corrente, opportunamente incrementato.

Nel caso generale può essere specificato in vari modi:

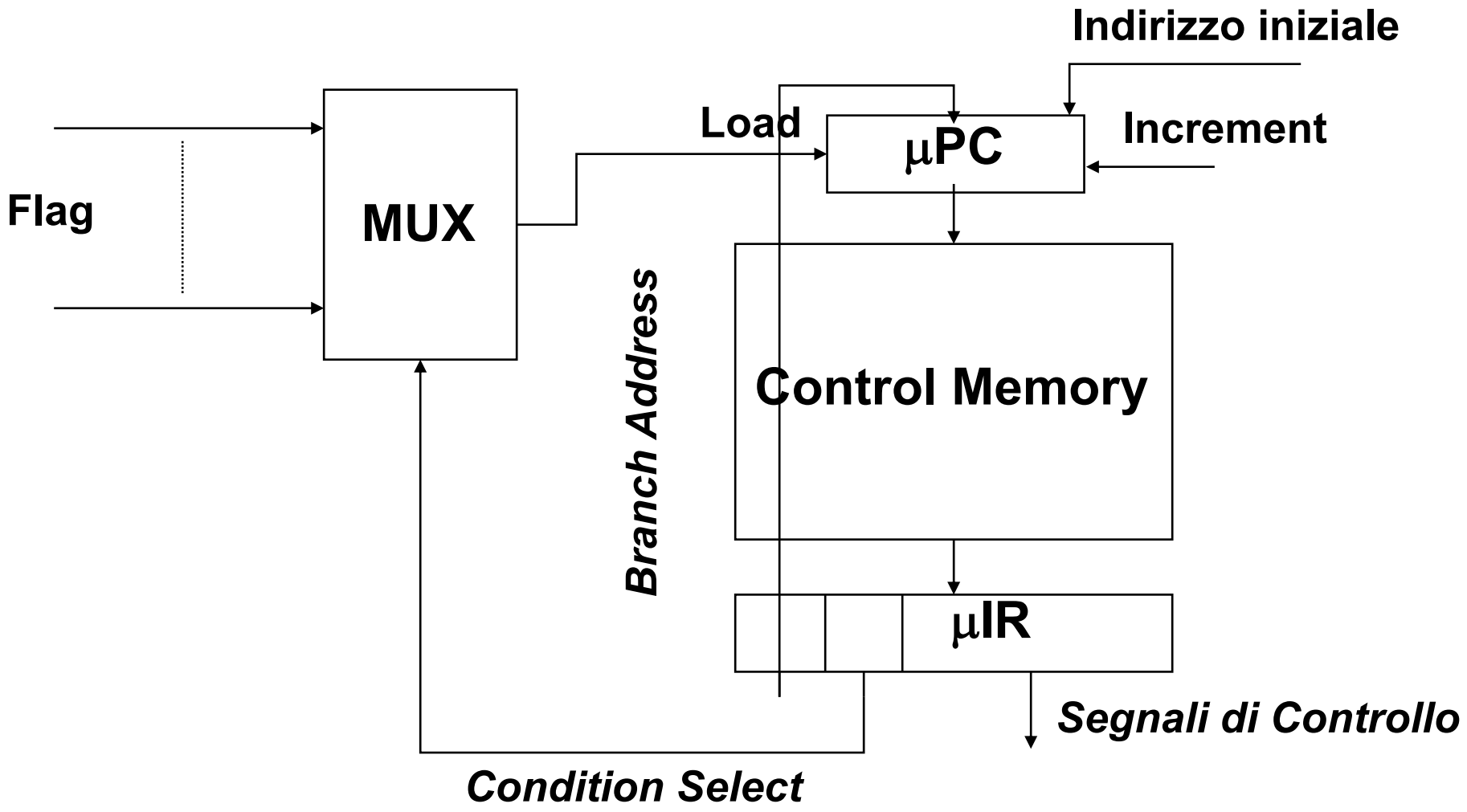
- **ogni microistruzione ha un campo aggiuntivo, che viene utilizzato dalle microistruzioni di salto per contenere l'indirizzo di salto, oppure**
- **le microistruzioni di salto hanno un formato diverso da quello delle microistruzioni normali.**

Esempio

La microistruzione è costituita da 3 campi:

- **Condition Select: specifica quali segnali esterni (di condizione) considerare**
- **Branch Address: indirizzo a cui saltare se la condizione selezionata è soddisfatta**
- **Control Fields: segnali di controllo da attivare.**

Esempio (Cont.)



Microprogrammazione orizzontale

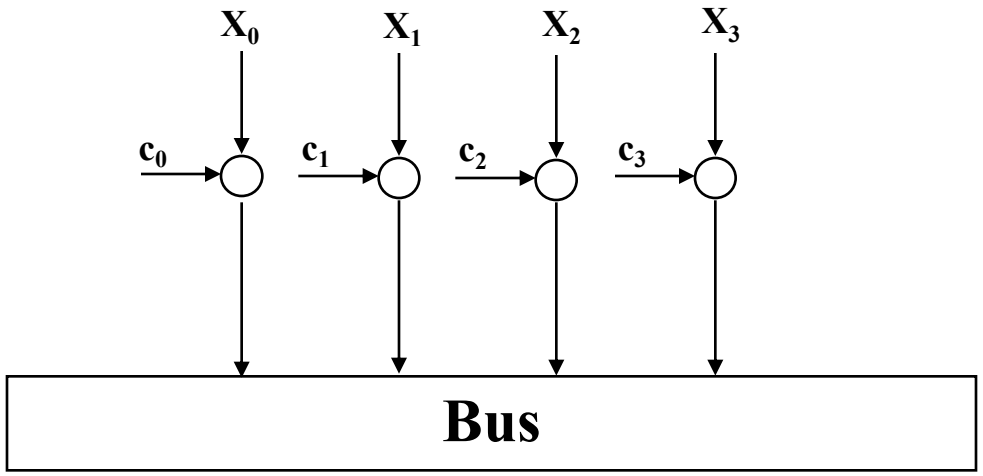
Nel caso più semplice (microprogrammazione *orizzontale*), le microistruzioni contengono un bit per ogni segnale di controllo.

Si ottiene così massima velocità di esecuzione: i segnali di controllo sono pilotati direttamente, senza bisogno di manipolazioni.

Esistono tuttavia alcune controindicazioni:

- **la lunghezza delle microistruzioni può divenire eccessiva**
- **talune combinazioni di valori dei segnali di controllo possono non verificarsi mai.**

Esempio



Configurazioni possibili:

0 0 0 1

0 0 1 0

0 1 0 0

1 0 0 0

Microprogrammazione verticale

In questo caso ogni microistruzione memorizza in maniera codificata le informazioni da inviare ai segnali di controllo.

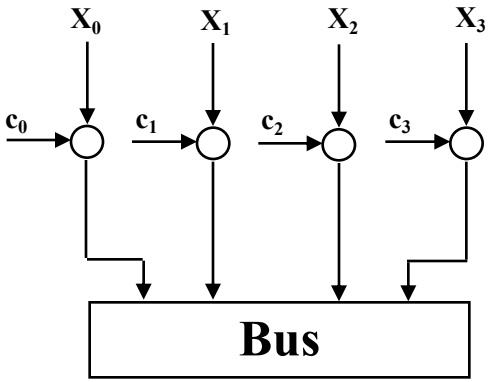
I segnali di controllo sono suddivisi in gruppi.

Ogni gruppo di segnali di controllo corrisponde ad un gruppo di bit (*Control Field*) della singola microistruzione, il cui valore è codificato.

Il valore di ciascun segnale di controllo viene quindi prodotto dalle uscite di un decoder, pilotato dai bit della microistruzione.

In questo modo le microistruzioni hanno quindi una lunghezza più ridotta, in quanto per l'i-esimo gruppo (composto da n_i segnali) si memorizzano $\log_2 n_i$ bit.

Esempio



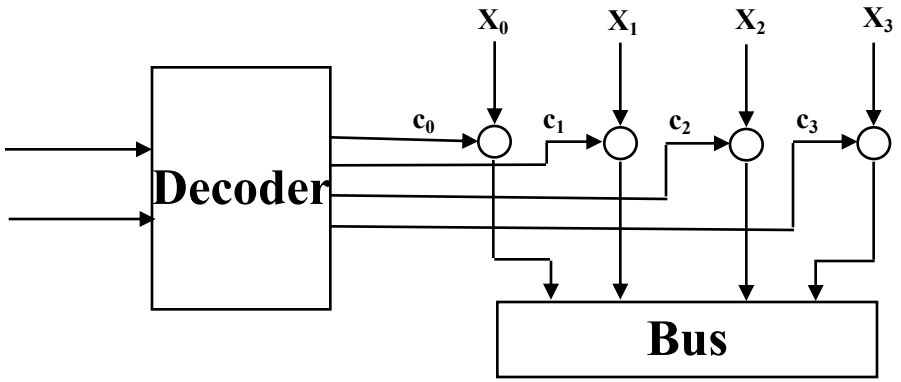
Configurazioni possibili:

0 0 0 1

0 0 1 0

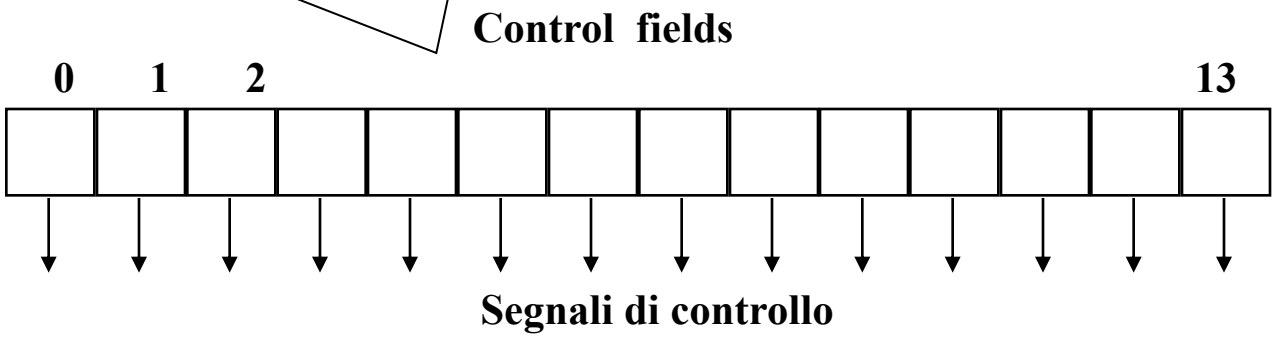
0 1 0 0

1 0 0 0

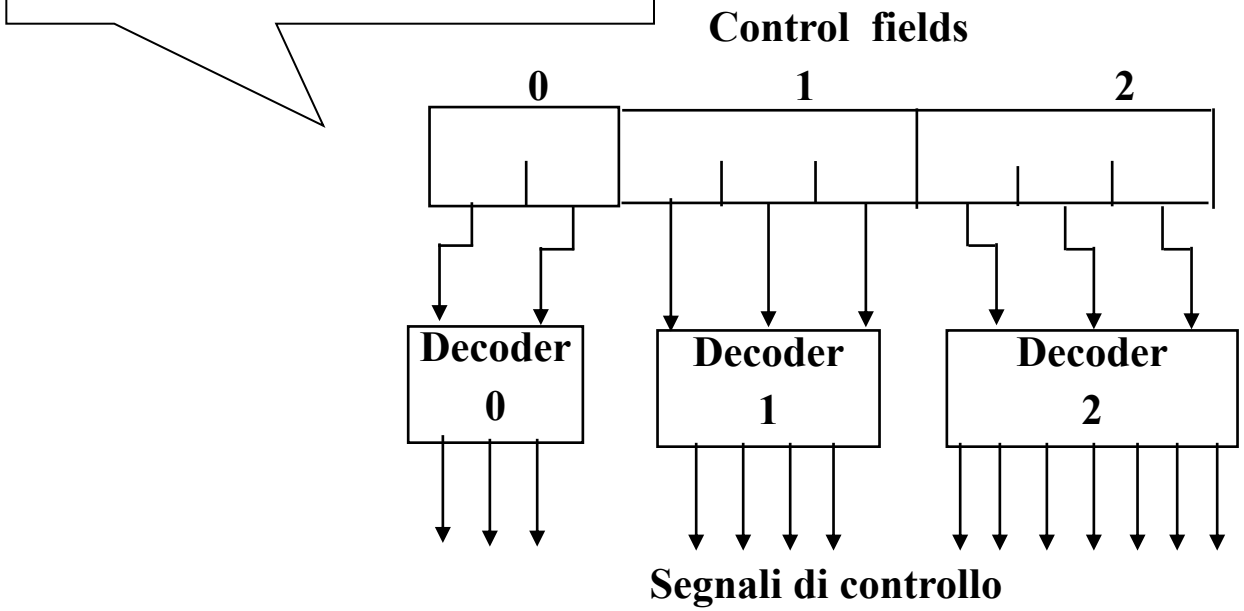


Alternative

Microprogrammazione orizzontale



Microprogrammazione verticale



Compatibilità

Due segnali di controllo sono *compatibili* se non sono mai attivati nella stessa microistruzione.

Essi possono dunque appartenere allo stesso gruppo (o classe di compatibilità).

Una *classe di compatibilità* è un insieme di segnali di controllo i quali sono a 2 a 2 compatibili.

Esempio

microistruzione	segnali di controllo							
I_1	a	b	c				g	
I_2	a		c		e			h
I_3	a			d		f		
I_4		b	c			f		

I segnali *a* e *f* non sono compatibili.

I segnali *e*, *f*, e *g* compongono una classe di compatibilità.

I segnali *d* e *e* sono compatibili.

Minimizzazione del microcodice

Per minimizzare il parallelismo della memoria di microcodice è necessario individuare la ripartizione dei segnali di controllo in classi di compatibilità che minimizza la funzione

$$\sum_{i=0}^{k} \log_2 n_i$$

dove K è il numero delle classi, ed n_i è il numero di segnali di controllo in ciascuna classe.

Esempi
di CPU microprogrammate

- **IBM System 360/370**
- **Intel 80x86**
- **Motorola 680x0.**

Caratteristiche delle UC microprogrammate

+ ***Flessibilità*: modificare un'istruzione o aggiungerne di nuove comporta semplicemente la modifica del contenuto della memoria di microprogramma**

- ***Velocità*: l'esecuzione di un'istruzione richiede una serie di accessi alla memoria di microprogramma; un'UC microprogrammata è quindi in genere più lenta di un'UC cablata**

- ***Costo*: la presenza della memoria di microprogramma e della relativa logica fa crescere il costo in termini di hardware.**