Le unità di controllo

M. Sonza Reorda

Politecnico di Torino
Dipartimento di Automatica e Informatica

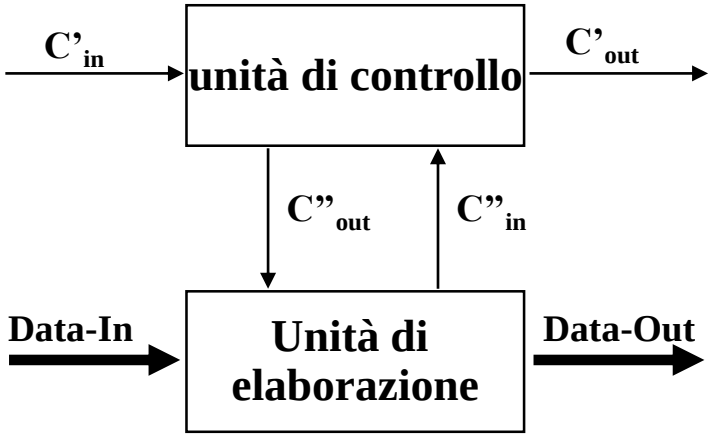
Sommario

- **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.

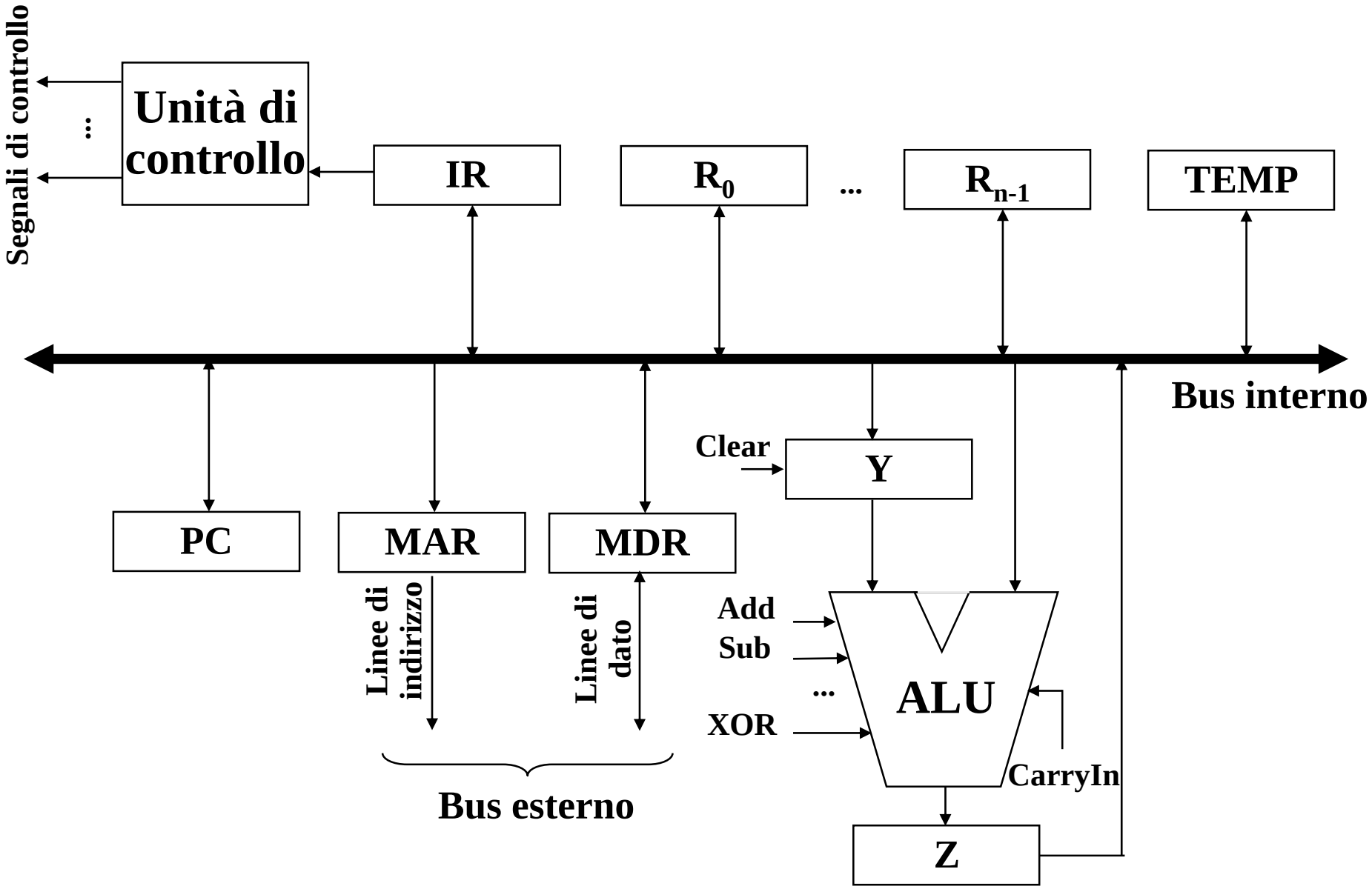
Operazioni elementari

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:

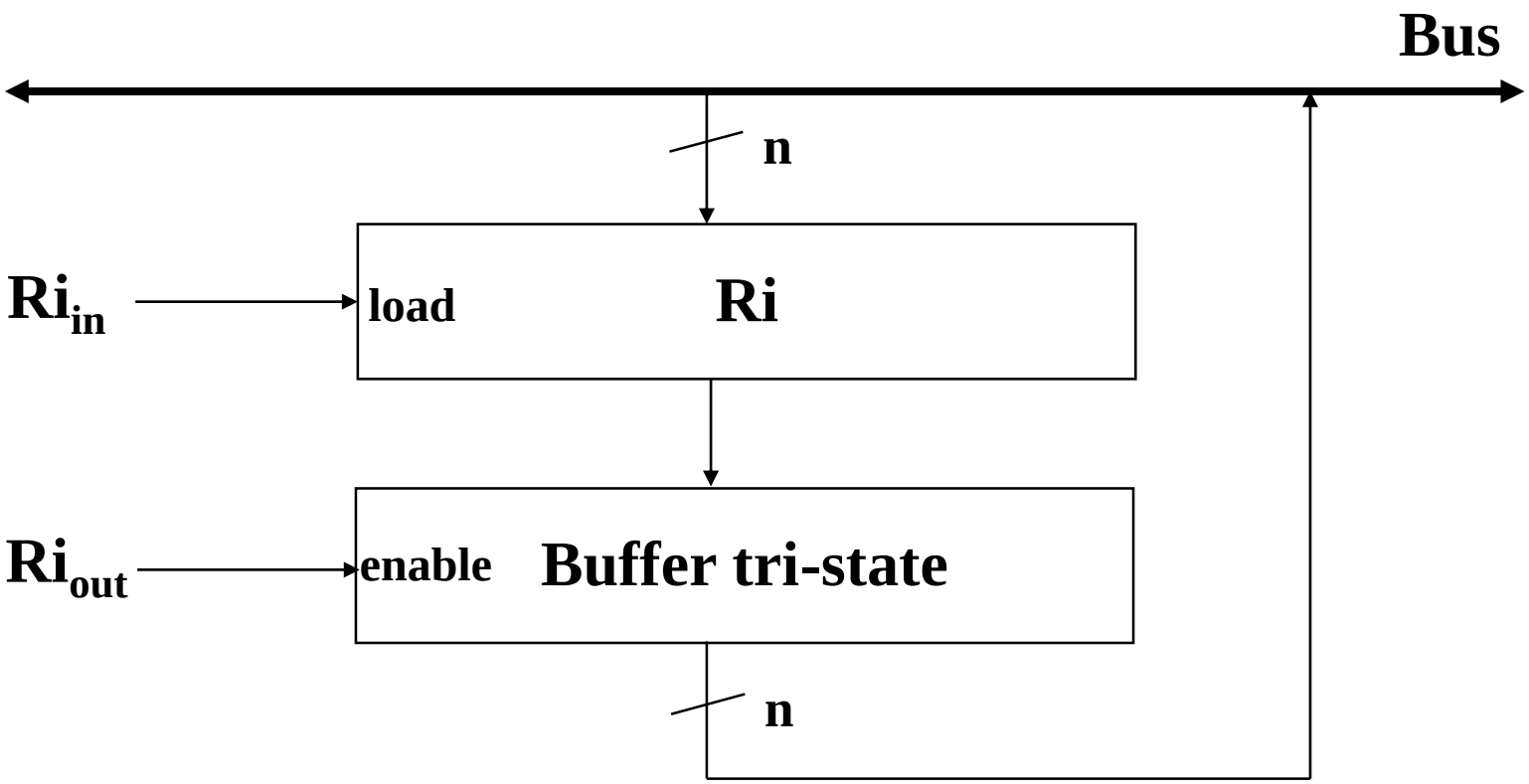
- **Ri_{in}, che fa sì che il registro carichi il valore presente sul bus**
- **Ri_{out}, che fa sì che il contenuto del registro venga trasferito sul bus; quando il segnale non è attivo, il registro forza sul bus lo stato di *alta impedenza* (circuito aperto).**

Vincoli

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

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

Realizzazione dell'interfaccia



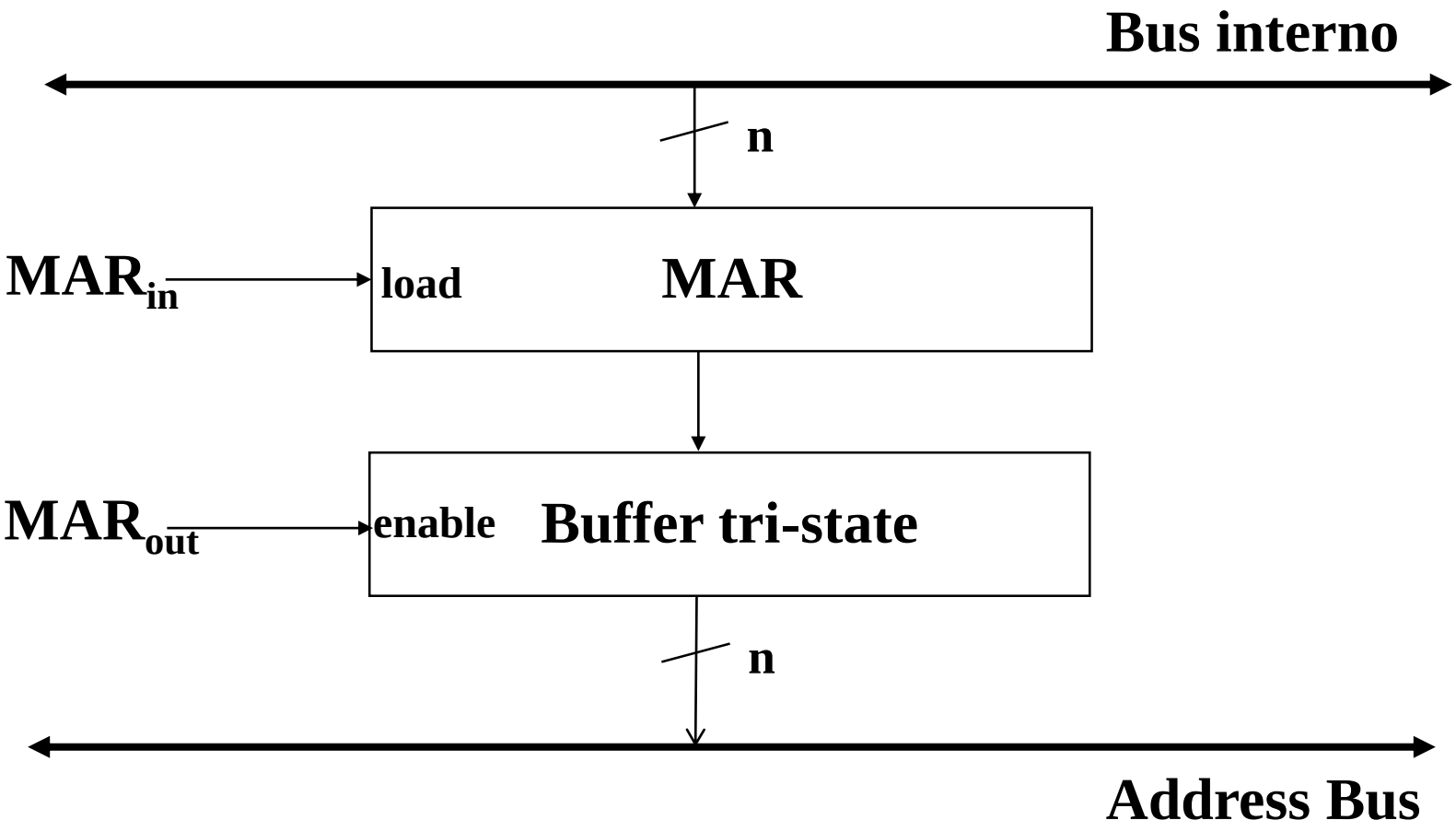
Trasferimenti tra registri

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

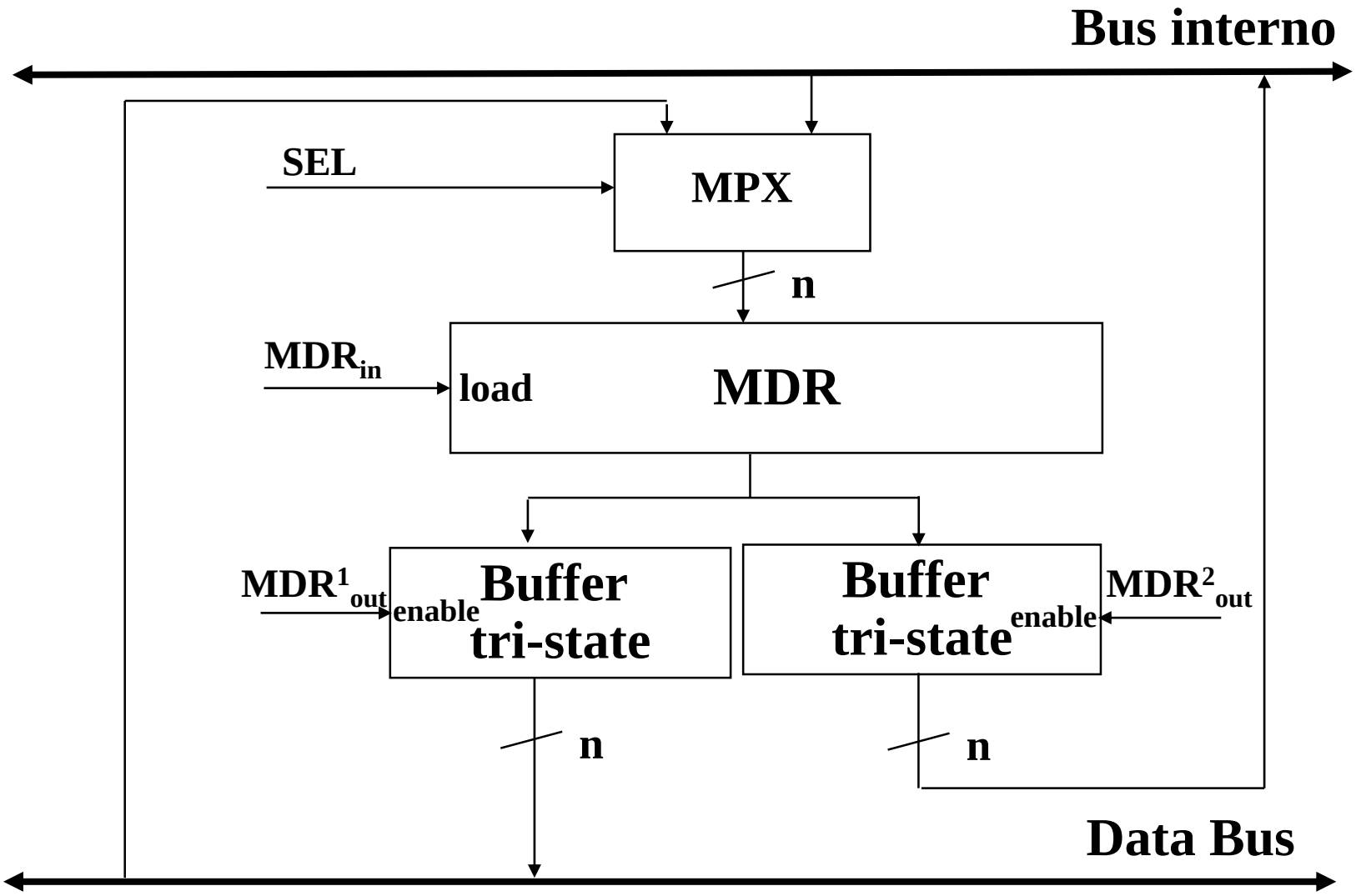
Ad esempio, per trasferire il contenuto del registro R1 in R4 si deve:

- **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.**

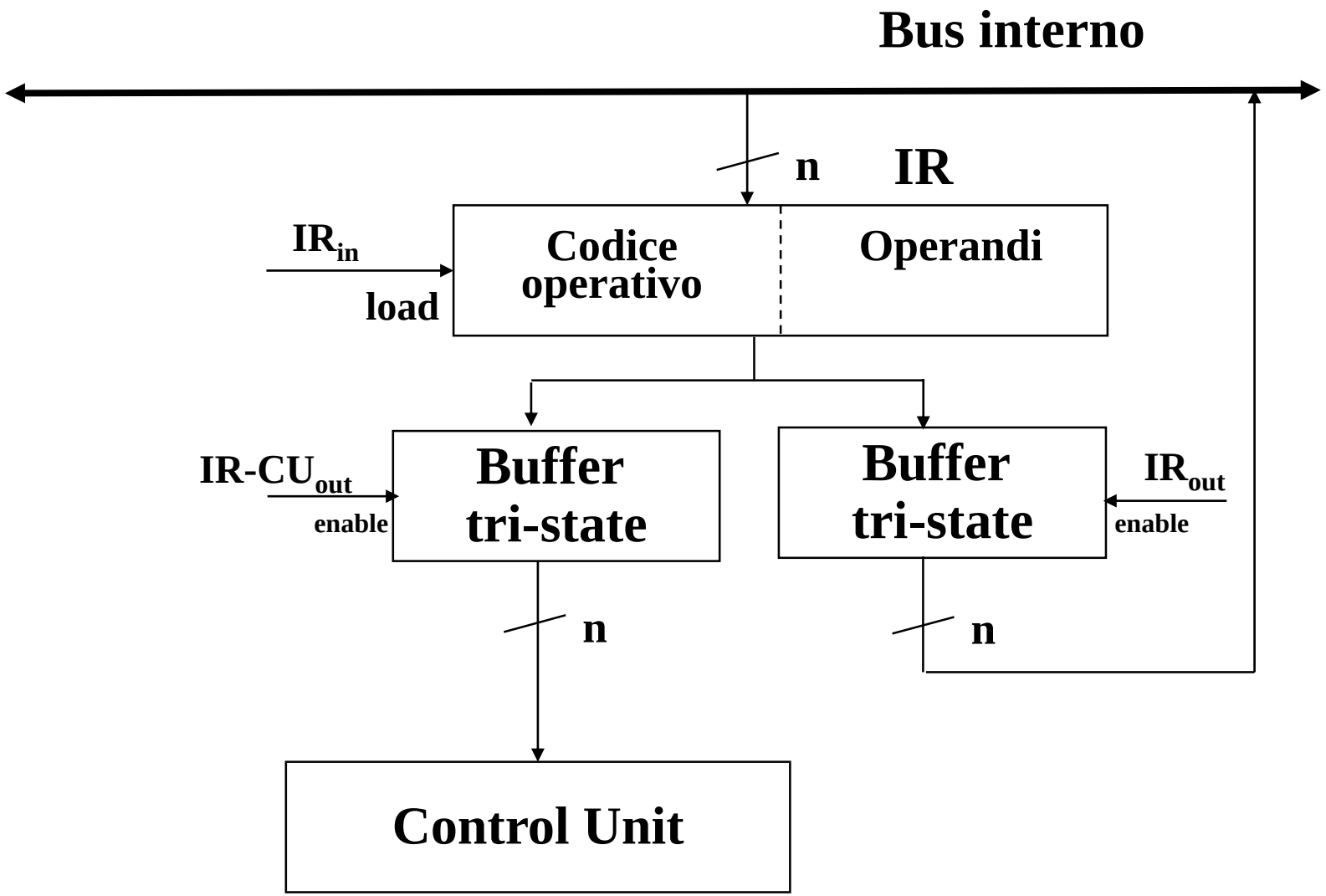
Connessione di MAR



Connessione di MDR



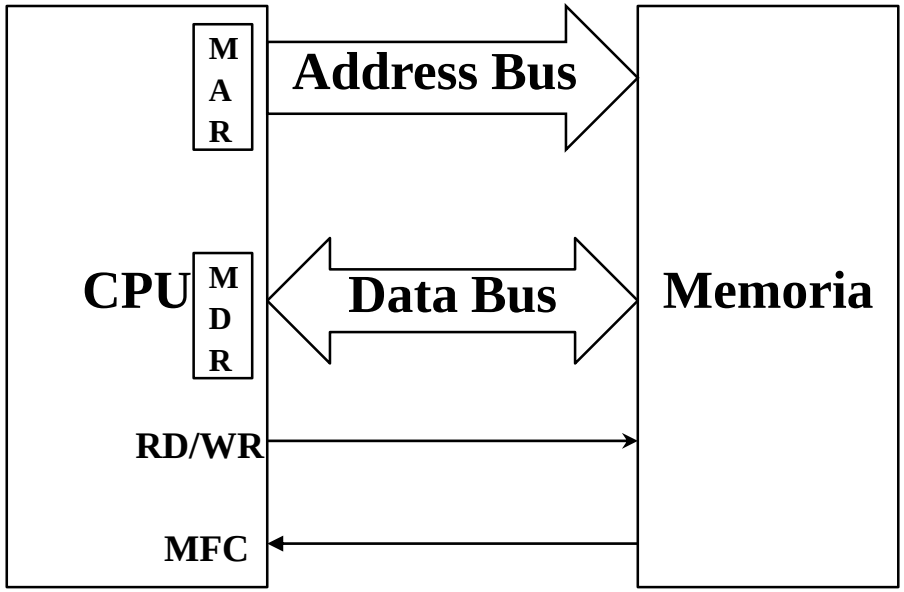
Connessione di IR



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 e mettere sul bus il dato in lettura)**
- **una volta completats 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

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

Prelievo di una parola dalla memoria: segnali di controllo

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

Segnali di Controllo abilitati

- **$R1_{out}$, MAR_{in}**
- **MAR_{out}, RD**
- **Aspetta MFC, SEL (Bus Esterno), MDR_{in}**
- **MDR^{2}_{out}, $R2_{in}$**

Fetch

- **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**
- **Decodifica dell'Istruzione**

Fetch

- **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**
- **Decodifica dell'Istruzione**

Segnali di controllo abilitati:

- **PC_{out}, MAR_{in}, Clear Y, Set Carry In to ALU, Add, Z_{in}**
- **MAR_{out}, RD, Z_{out}, PC_{in}**
- **Aspetta MFC, SEL (Bus Esterno), MDR_{in}**
- **MDR^{2}_{out}, IR_{in}**
- **IR-CU_{out}**

Esecuzione di un'operazione logica o aritmetica

Per eseguire la somma tra i registri R1 ed R2 e mettere il risultato in R3 si devono eseguire le seguenti microistruzioni, di cui si riportano i segnali di controllo attivati:

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

Esecuzione di un'operazione logica o aritmetica

Per eseguire la somma tra i registri R1 ed R2 e mettere il risultato in R3 si devono eseguire le seguenti microistruzioni, di cui si riportano i segnali di controllo attivati:

- **$Y \leftarrow R1$**
- **$Z \leftarrow Y + R2$**
- **$R3 \leftarrow Z$**

Con i corrispondenti seguenti segnali di controllo:

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

Esempio

Si consideri l'istruzione

Add [R3], R1

Essa somma a R1 il contenuto della cella di memoria il cui indirizzo è contenuto in R3, e mette il risultato in R1.

- **MAR ← R3**
- **Attiva il segnale di lettura, Y ← R1**
- **Aspetta sino al segnale MFC, MDR ← Bus esterno**
- **Z ← MDR + Y**
- **R1 ← Z, End**

Esempio

Add [R3], R1

- **MAR ← R3**
- **Attiva il segnale di lettura, Y ← R1**
- **Aspetta sino al segnale MFC, MDR ← Bus esterno**
- **Z ← MDR + Y**
- **R1 ← Z**

I segnali di controllo sono i seguenti:

- **$R3_{out}$, MAR_{in}**
- **MAR_{out}, RD, $R1_{out}$, Y_{in}**
- **Aspetta MFC, SEL (Bus Esterno), MDR_{in}**
- **MDR^2_{out}, Add, Z_{in}**
- **Z_{out}, $R1_{in}$, End**

Istruzioni di salto

Le microistruzioni che devono essere eseguite in corrispondenza di una istruzione di salto incondizionato sono:

- **Y ← PC, Z ←(Campo operando di IR)+Y**
- **PC ← Z**

- **Nota: si suppone che l'operando specifichi l'offset rispetto all'istruzione corrente.**

Istruzioni di salto

Le microistruzioni che devono essere eseguite in corrispondenza di una istruzione di salto incondizionato sono:

- **Y ← PC**
- **Z ←(Campo operando di IR) + Y**
- **PC ← Z**

I segnali che devono essere attivati per eseguire un'istruzione di salto incondizionato sono:

- **PC_{out}, Y_{in}**
- **(Campo operando di IR)$_{out}$, $Z_{in,}$ Add**
- **Z_{out}, PC_{in}, End**

Salto condizionato

JZ lab

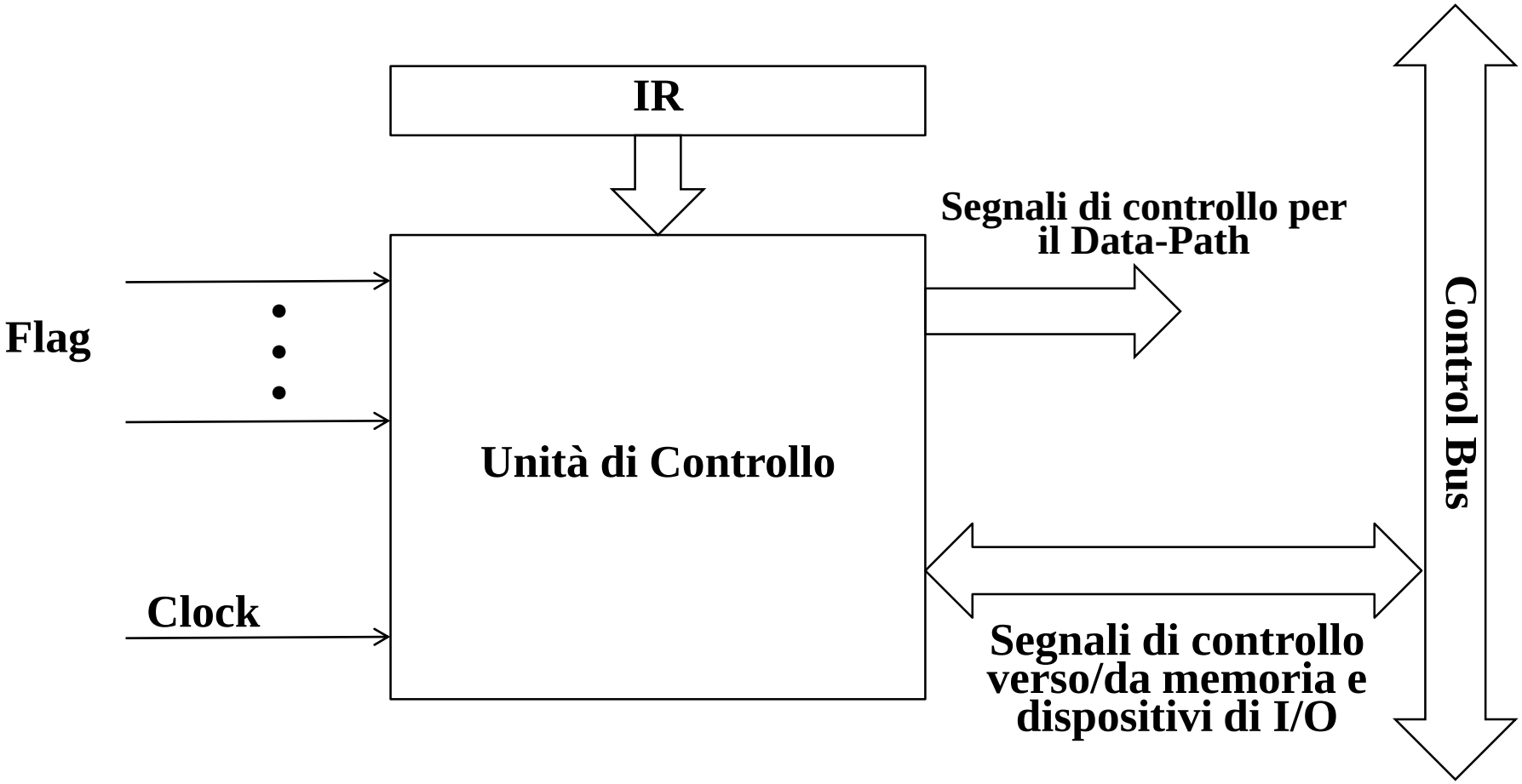
Le microistruzioni che devono essere eseguite sono le seguenti:

- **Y ← PC, If ZF = 1 then**
  - **Z ←(Campo operando di IR) + Y**
  - **PC ← Z**

I segnali che devono essere attivati per eseguire un'istruzione di salto condizionato sono (nell'ordine):

- **PC_{out}, Y_{in}, IF ZF = 0 then End Else**
  - **(Campo operando di IR)$_{out}$, Add, Z_{in}**
  - **Z_{out}, PC_{in}, End**

Progetto dell'Unità di Controllo



Progetto dell'unità di controllo

Si parte generalmente da una descrizione del suo funzionamento, basata ad esempio 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).

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.

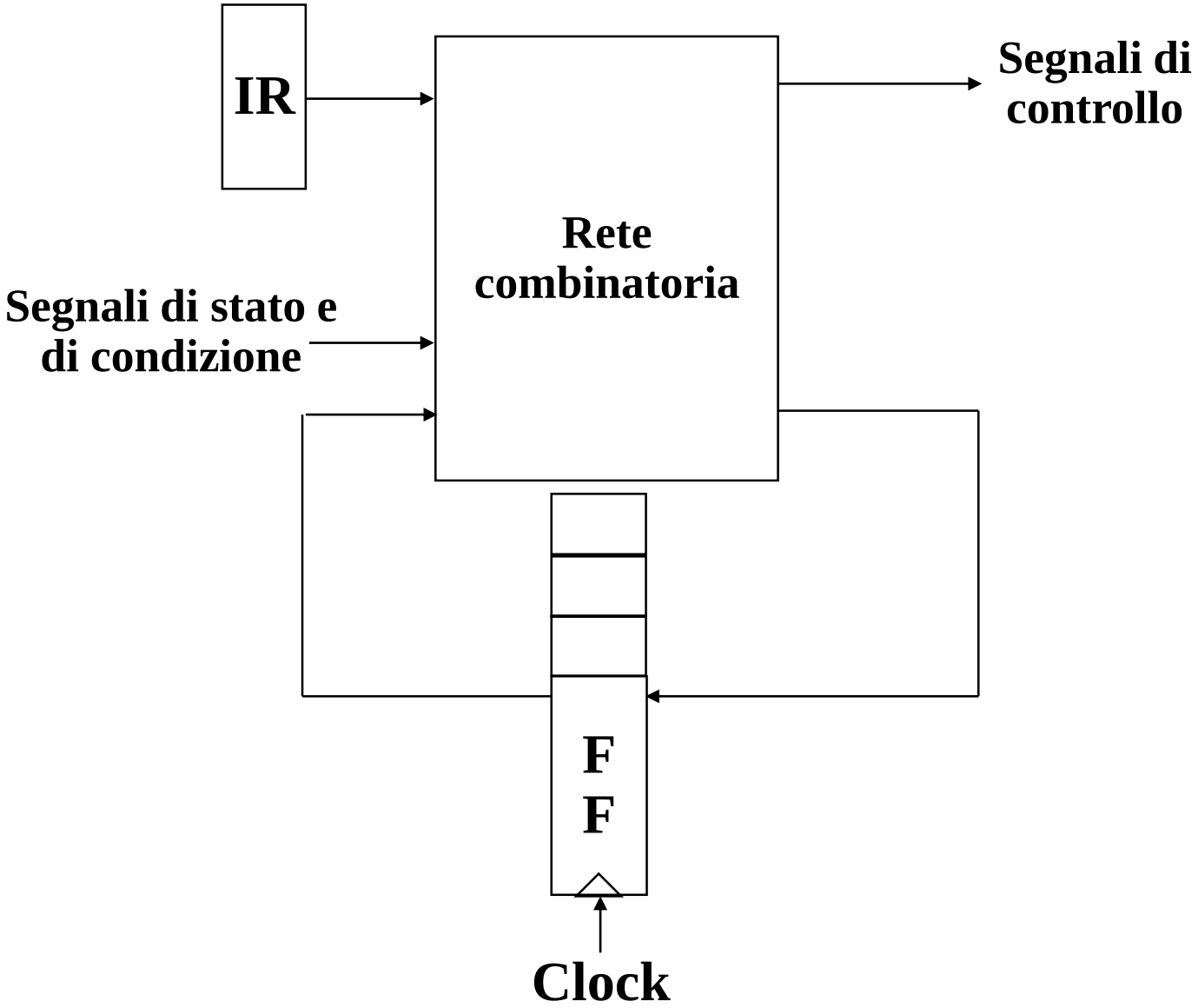
Si costruisce così una tabella degli stati, che viene poi trattata con i normali metodi di sintesi, e da essa si genera l'hardware corrispondente.

Complessità dell'unità di controllo

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

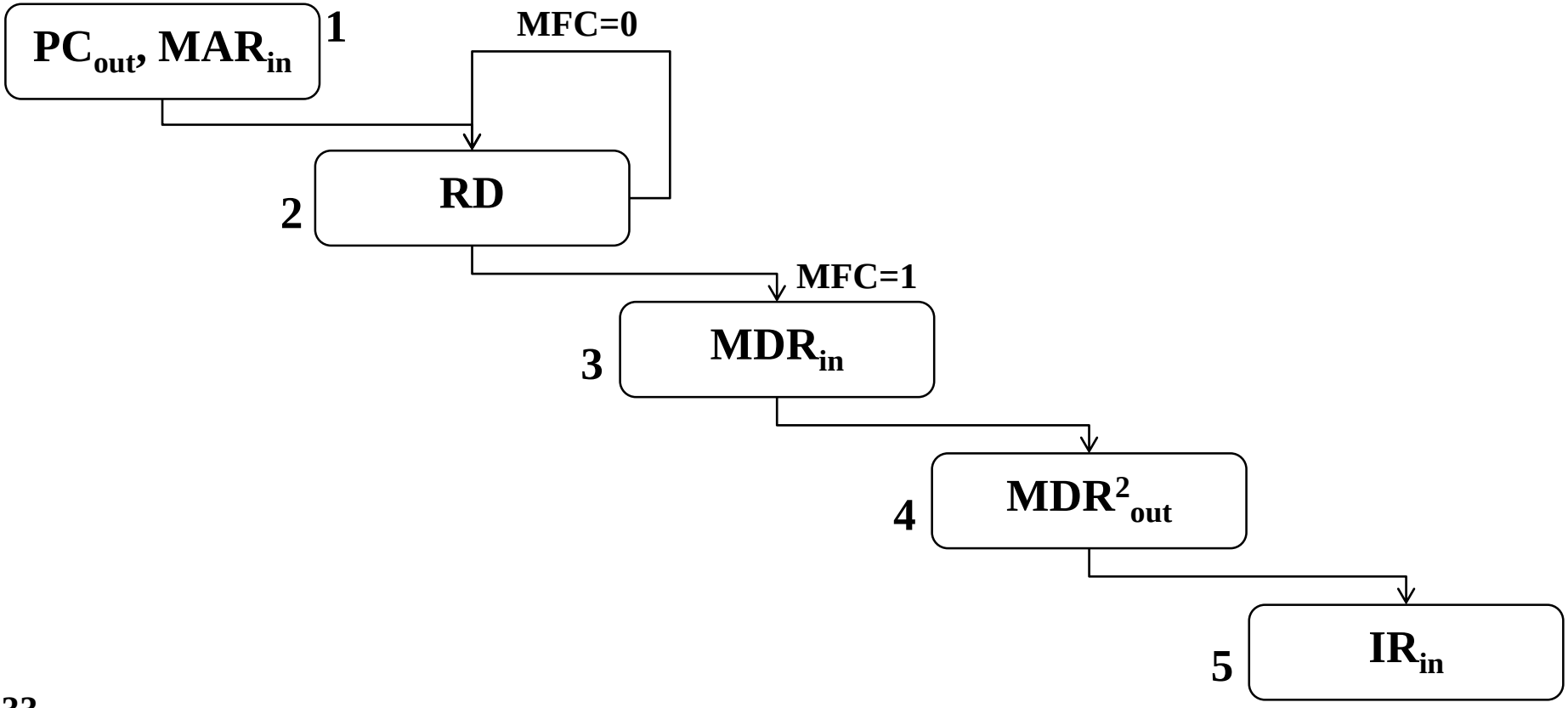
- **il numero di segnali di controllo (provenienti dall'esterno e dall'unità di elaborazione)**
- **il numero di stati.**

Architettura



Esempio

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

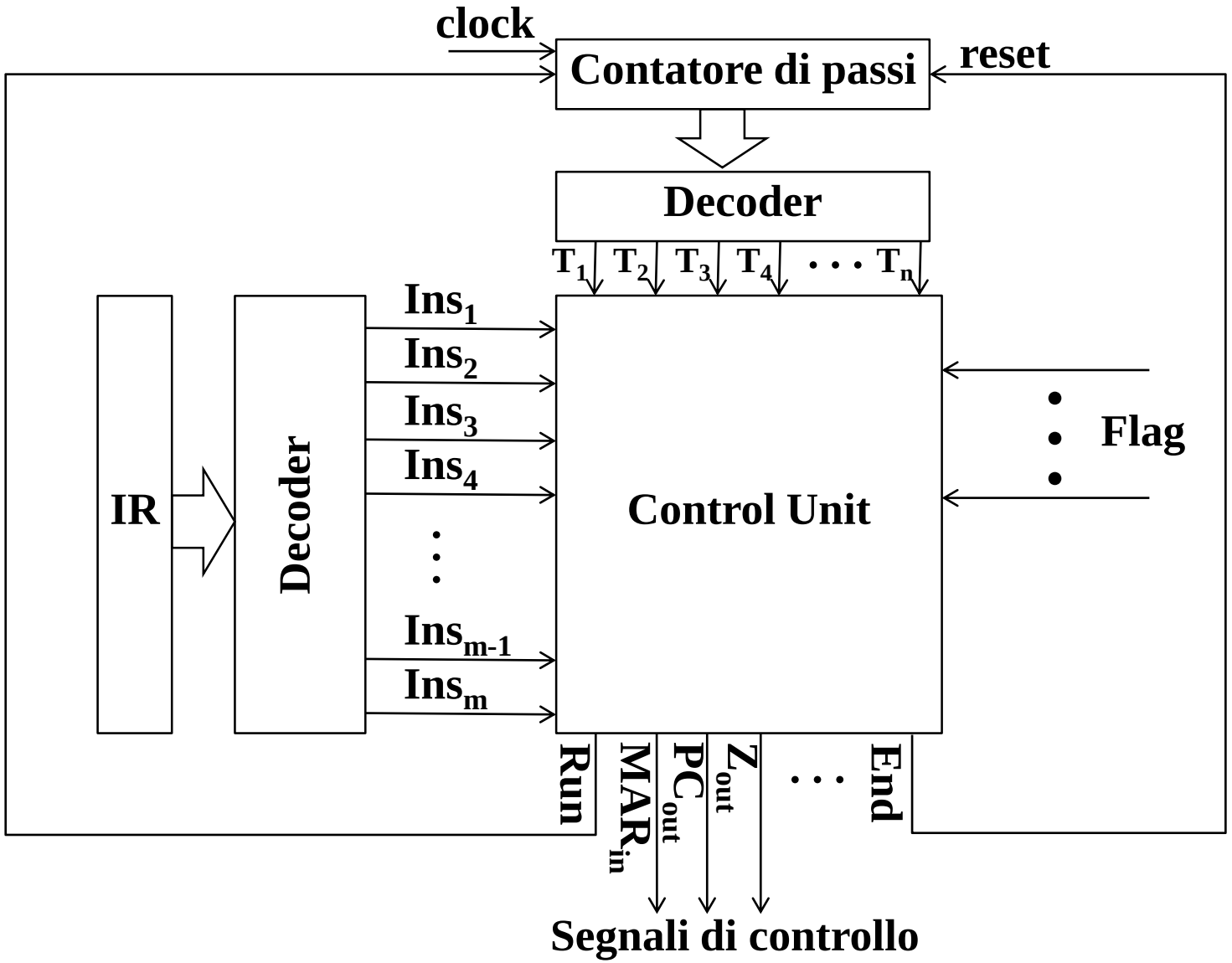


Esempio

La corrispondente tabella degli stati è:

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

Esempio di realizzazione



Esempio

Si consideri il segnale di controllo MAR_{in}, per caricare il registro MAR. Esso viene attivato in corrispondenza delle seguente microperazioni:

- **Fetch (tempo T_1)**
- **Primo ciclo (tempo T_n) della fase di execute, per la lettura (load) di un operando per le istruzioni con operando indiretto, ad es. ADD [R1], R2**
- **Ciclo avanzato (tempo T_m) della fase di execute, per la scrittura (store) del risultato dell'istruzione, MOV R1, [R3]**

$$MAR_{in} = T_1 + T_n \bullet (ADD\_IND + XOR\_IND + MOV\_IND + ...) + T_m \bullet ADD\_STORE + XOR\_STORE + MOV\_STORE + ...)$$

Hardwired Control Unit: Vantaggi

- **Minimizza l’area di silicio utilizzata**
- **Elevata efficienza in termini di velocità delle istruzioni**
- **Permette di minimizzare il costo del circuito**

Hardwired Control Unit: Svantaggi

- **Le dimensioni della macchina a stati finiti può essere tale da renderla intrattabile**
- **Alcune informazioni legate al funzionamento dell'unità di controllo (ad es. la presenza di cicli) vengono perse**
- **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**
- **L'unità di controllo dell’8088.**

Introduzione

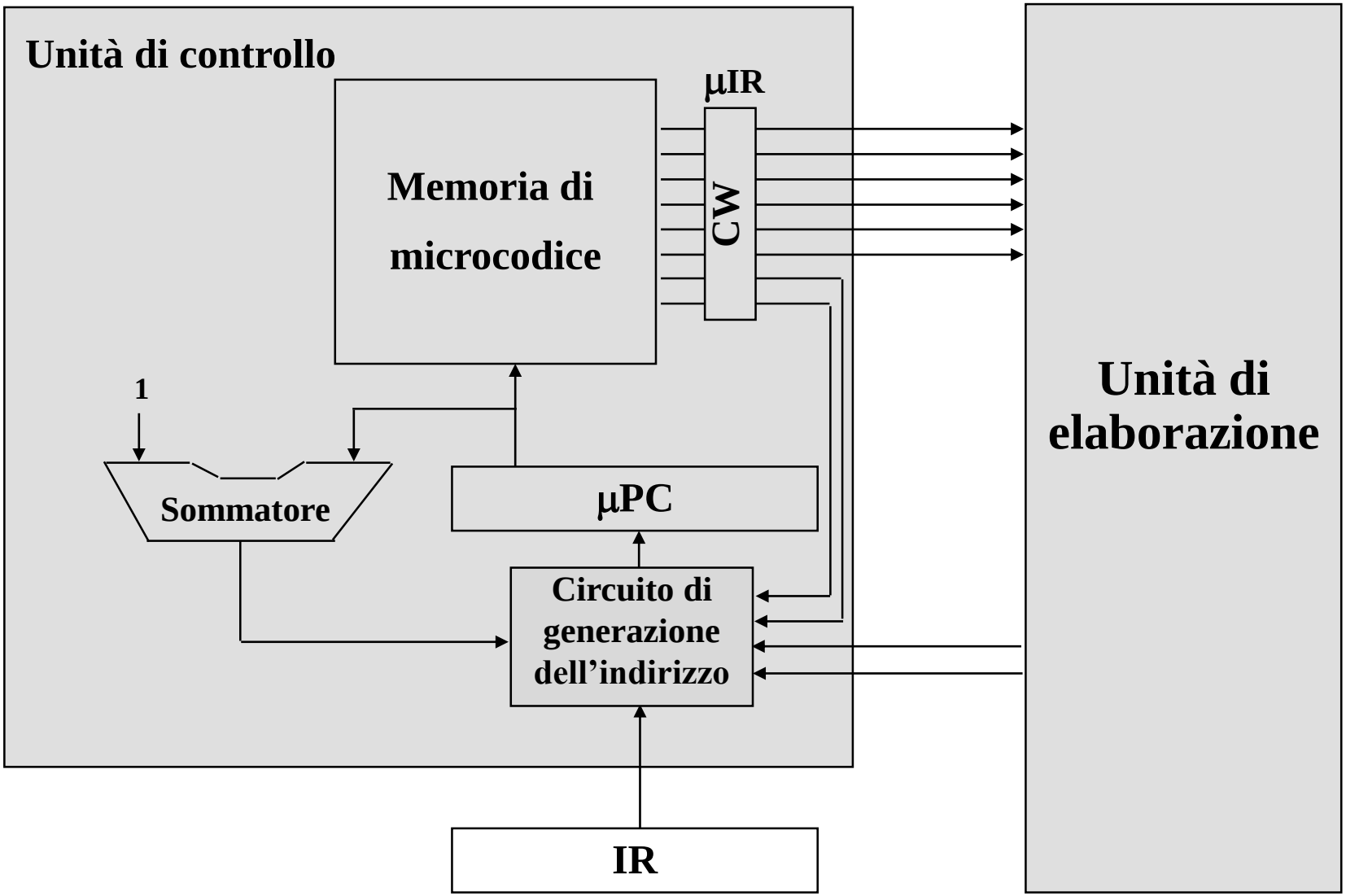
La microprogrammazione fu proposta per la prima volta da M.V. Wilkes nel 1951 (articolo "The Best Way to Design an Automatic Calculating Machine").

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.

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:

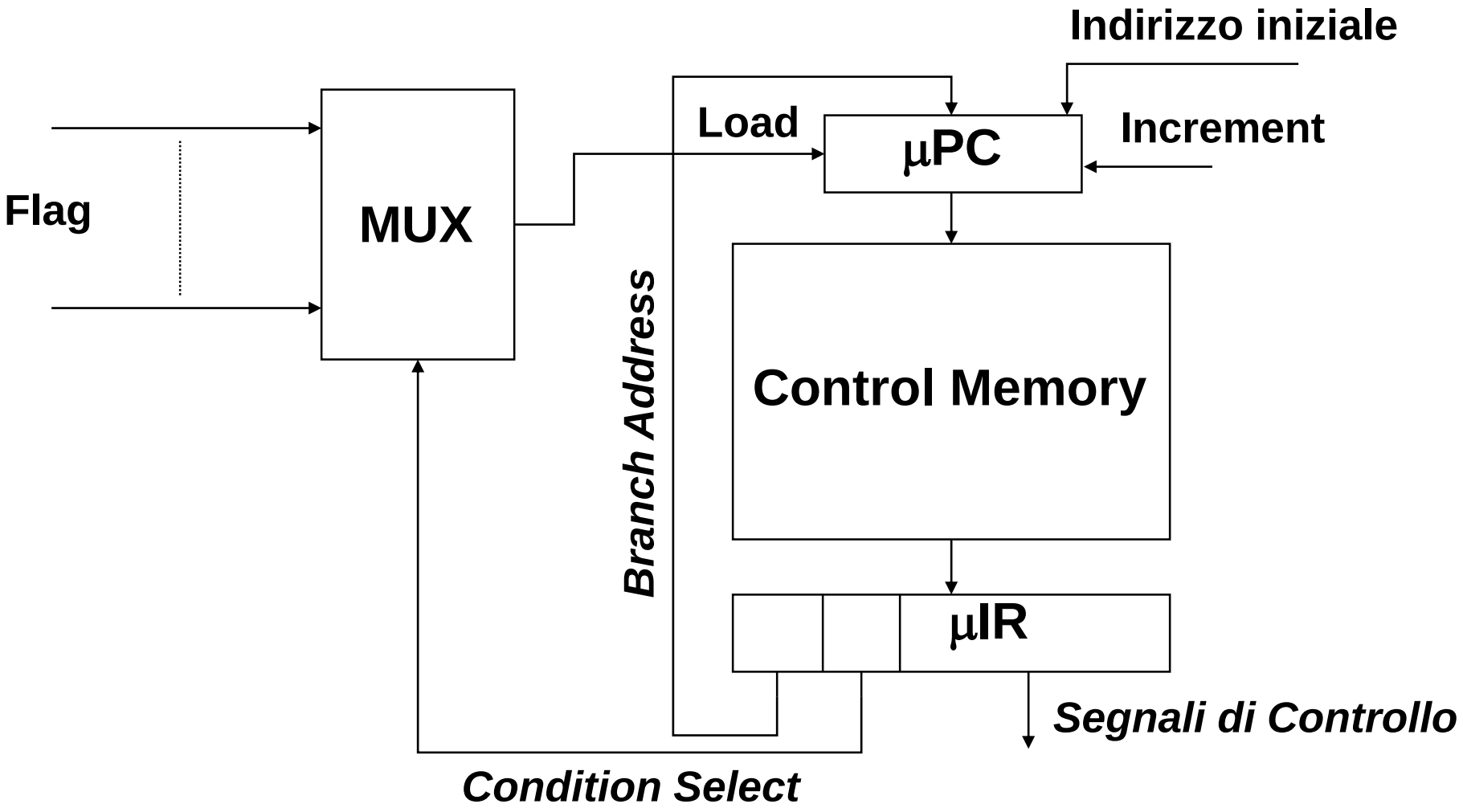
- **ogni microistruzione ha un campo aggiuntivo, che viene utilizzato dalle microistruzioni di salto condizionato per contenere il possibile indirizzo di salto, oppure**
- **le microistruzioni di salto hanno un formato diverso da quello delle microistruzioni normali.**

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

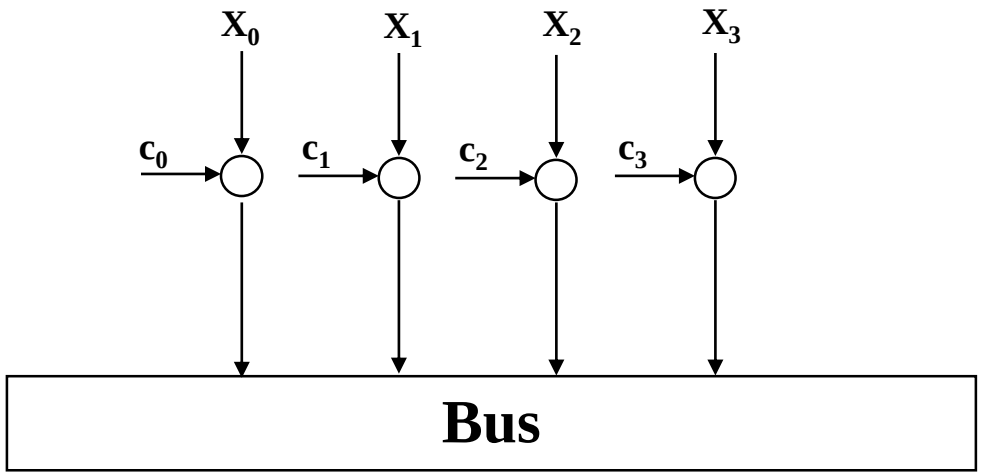
Nel caso più semplice (microprogrammazione *orizzontale*), le microistruzioni contengono un bit per ogni segnale di controllo.

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.

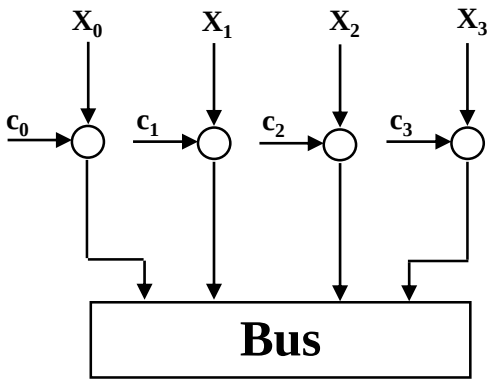
I segnali di controllo sono suddivisi in gruppi.

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



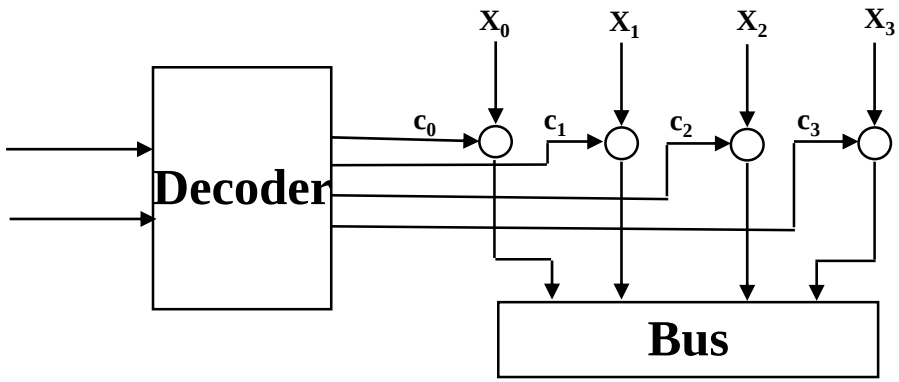
Configurazioni possibili:

0 0 0 1

0 0 1 0

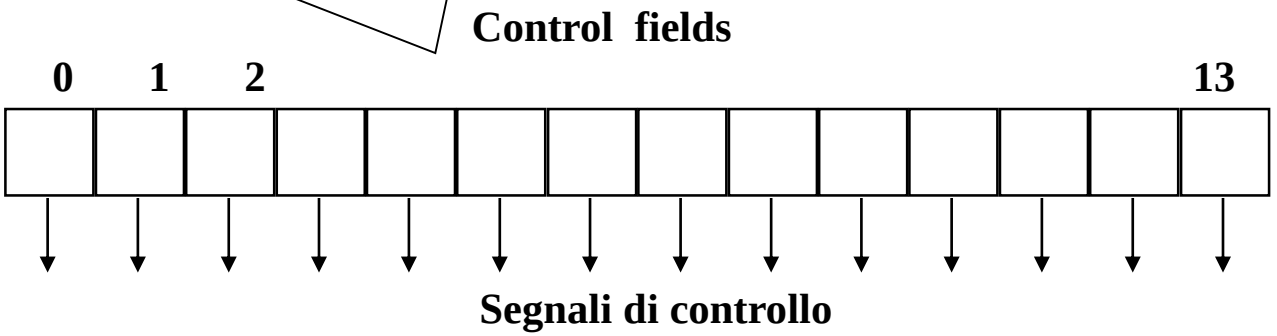
0 1 0 0

1 0 0 0

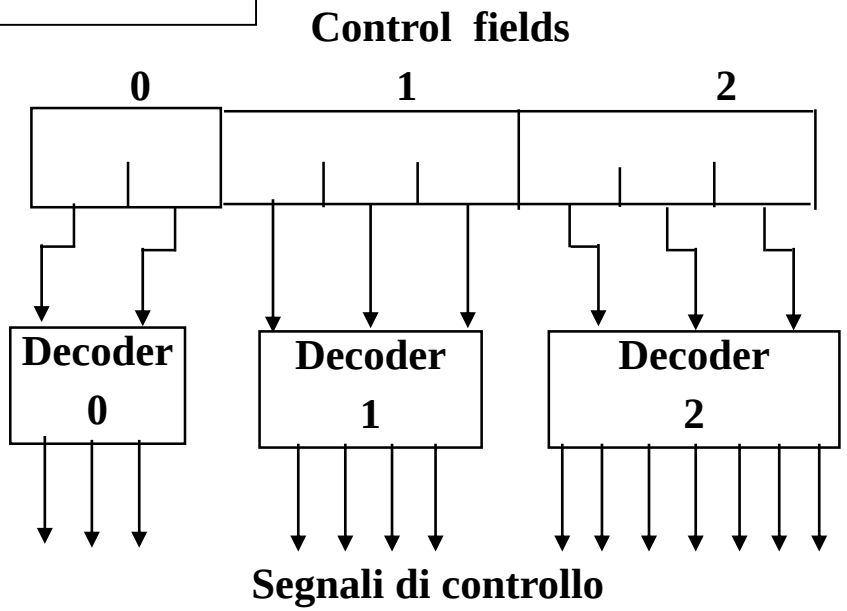


Alternative

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**

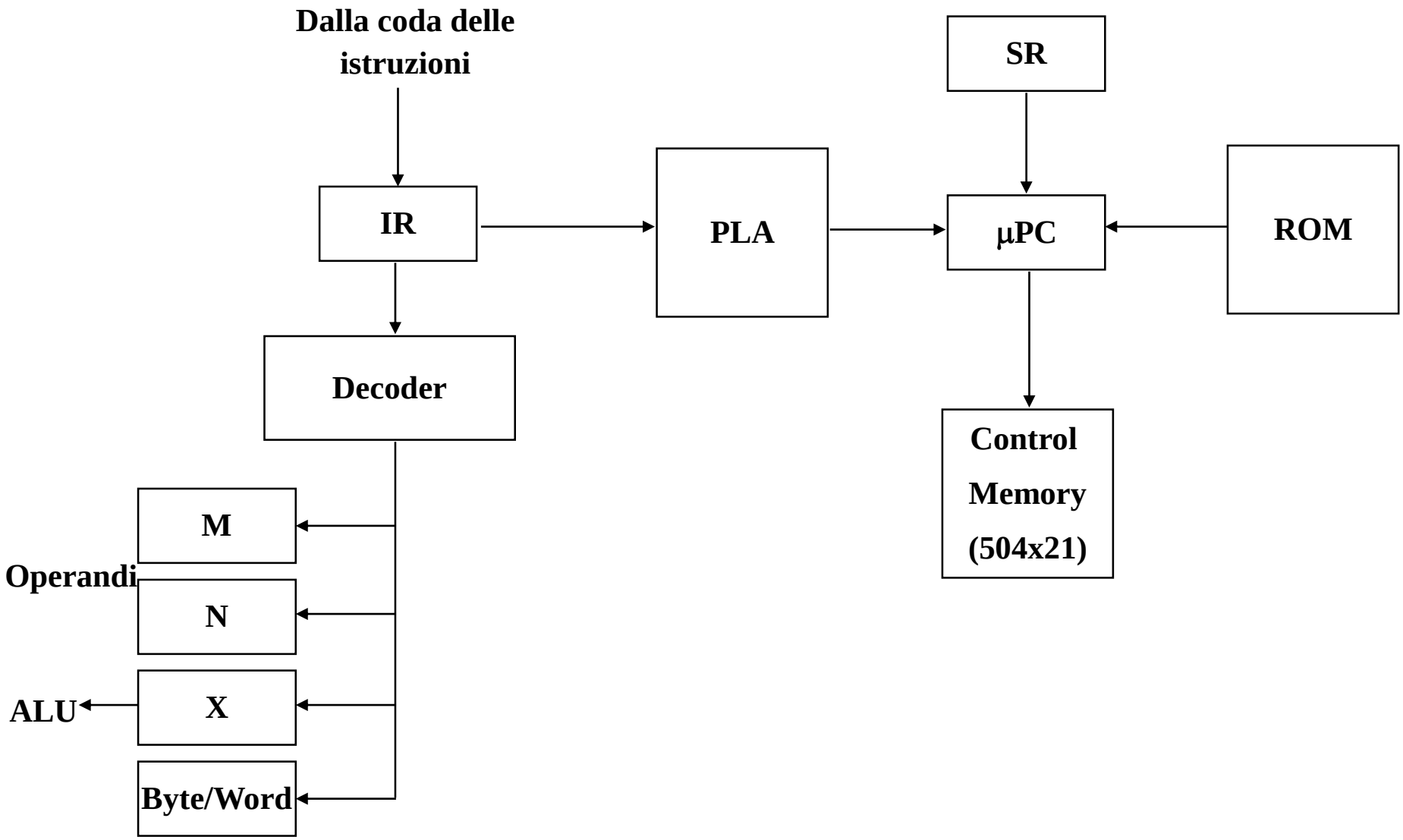
- ***Costo*: la presenza della memoria di microprogramma e della relativa logica fa crescere il costo in termini di hardware.**

L’unità di controllo dell’8088

Caratteristiche generali:

- **l’unità di controllo dell’8088 (come quella dell’8086) è in parte cablata ed in parte microprogrammata**
- **le microistruzioni hanno ampiezza pari a 21 bit**
- **il formato delle microistruzioni è verticale**
- **la memoria di microcodice contiene 504 microistruzioni.**

Unità di controllo dell'8088



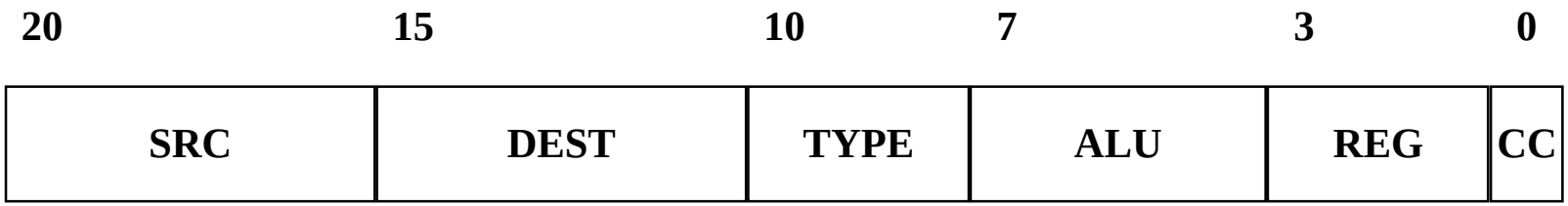
Funzionamento

- **Il modulo PLA converte il codice operativo dell'istruzione nell'IR nell'indirizzo della prima microistruzione corrispondente.**
- **Il microcodice è diviso in *burst*, composti al più da 16 microistruzioni: ogni burst corrisponde ad una istruzione, oppure ad una operazione generale (ad esempio il calcolo degli indirizzi).**
- **Esistono 2 tipi di salti:**
  - **quello *breve*, all'interno del burst**
  - **quello *lungo*, per saltare ovunque.**
- **Sono supportate le microprocedure, il cui indirizzo di ritorno viene salvato nel registro SR (*Subroutine Return*).**

Istruzioni di calcolo

- **Le istruzioni di calcolo corrispondono a 2 fasi:**
  - ***calcolo degli indirizzi degli operandi*** **(posti nei registri M e N) e** ***decodifica dell'operazione*** **che la ALU deve eseguire (nel registro X)**
  - ***esecuzione*** **del calcolo**
- **La stessa microprocedura può quindi eseguire operazioni diverse, a seconda dei valori presenti nei registri M, N ed X.**

Formato delle microstruzioni



- La parte sinistra permette il trasferimento tra registri
- La parte destra è codificata verticalmente
- TYPE seleziona il tipo di microistruzione:
  - operazione della ALU
  - operazione sulla memoria
  - salto breve/salto lungo
  - chiamata di microprocedura
  - prenotazione.

Formato delle microstruzioni

- **Il campo ALU può specificare una funzione, oppure indicare quella specificata dal registro X**
- **REG specifica gli operandi**
- **CC specifica se si devono settare i flag**
- **I salti lunghi ed i salti a procedura possono essere fatti solo ad uno tra 32 indirizzi predefiniti, identificabili tramite 5 bit: da questi la ROM produce l'indirizzo destinazione.**