

---

# Calcolatori Elettronici

## Parte Operativa e Parte Controllo

---

Ing. Gestionale e delle Telecomunicazioni  
A.A. 2008/09  
Gabriele Cecchetti

---

## Reti Sequenziali Sincrone

- Sommario:
  - Unità con Parte Operativa e Parte Controllo
- Riferimenti
  - G. Corsini “Dalle porte AND OR NOT al sistema calcolatore: un viaggio nel mondo delle reti logiche”: cap. “Reti Sequenziali Sincrone”

## Formalizzazione delle RSS di Mealy ritardato tramite programmi (1/2)

```
L0: OUTR:=0; goto {if <x1 x0> eq 11 then L1
  else L0}
```

```
L1: OUTR:=0; goto {if <x1 x0> eq 01 then L2
  else if <x1 x0> eq 11 then L1 else L0}
```

```
L2: OUTR:=if <x1 x0> eq 10 then 1 else 0; goto
  {if <x1 x0> eq 11 then L1 else L0}
```

Ove:  $L_h \Leftrightarrow S_h$  ed eseguire lo statement  $L_h$  significa:

- 1) calcolare sia lo stato di uscita (espressione a dx. di  $:=$  );
- 2) calcolare lo stato interno successivo (espressione a dx. di **goto**);
- 3) dopo il clock rinnovare OUTR e STAR.

## Formalizzazione delle RSS di Mealy ritardato tramite programmi (2/2)

Ogni *statement*  $k$ -esimo del programma è costituito da:

- Un *etichetta*  $L_k$  in corrispondenza biunivoca con lo stato interno ( $L_h \Leftrightarrow S_h$ )
- Una *microoperazione* che ha la forma  $OUTR=f^k(X)$
- Un *microsalto* che in generale ha la forma:  

```
goto {if conditionk1(X) then Lk1T else if ...
conditionkP(X) then LkPT else LkF}}
```
- L'insieme dei microsalti costituisce il *flusso di controllo*;
- L'insieme delle microoperazioni rappresenta il *flusso dei dati*.

## Rete combinatoria operativa

Sono presenti due microoperazioni:

```
OUTR:=0
```

```
OUTR:=if <x1 x0> eq 10 then 1 else 0
```

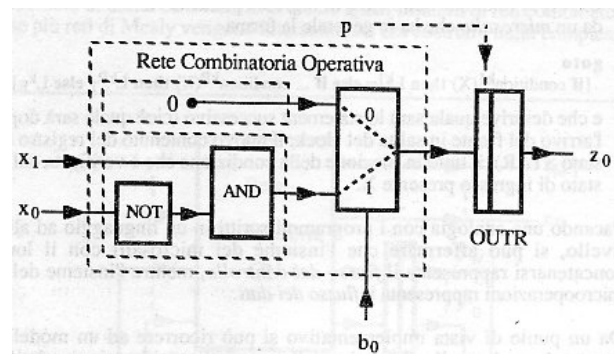
Esse possono essere compattate nella microoperazione globale:

```
OUTR:=case <b0> of
```

```
  { 0 : 0.
```

```
    1 : if <x1 x0> eq 10 then 1 else 0 }
```

Implementabile con il seguente  
circuito sequenziale  
(rete combinatoria OPERATIVA)  
dove b0 è la var. di comando.



## Rete combinatoria di condizionamento

b0 può essere generata da un RSS di Moore per mezzo del seguente programma.

```
L0: <b0>=0; goto {if <x1 x0> eq 11 then L1 else L0}
```

```
L1: <b0>=0; goto {if <x1 x0> eq 01 then L2 else if <x1  
x0> eq 11 then L1 else L0}
```

```
L2: <b0>=1; goto {if <x1 x0> eq 11 then L1 else L0}
```

Si noti che i microsalti possono essere di 2 tipi:

```
if <x1 x0> eq 11 then ...
```

```
if <x1 x0> eq 01 then ...
```

Allora è possibile definire una rete combinatoria di condizionamento avente in ingresso x1 ed x0 che genera le var. c1 e c0:

```
c1 = if <x1 x0> eq 11 then 1 else 0
```

```
c0 = if <x1 x0> eq 01 then 1 else 0
```

## Parte controllo

Il circuito che porta avanti il flusso di controllo è detto **parte controllo** ed è espresso quindi dal seguente programma:

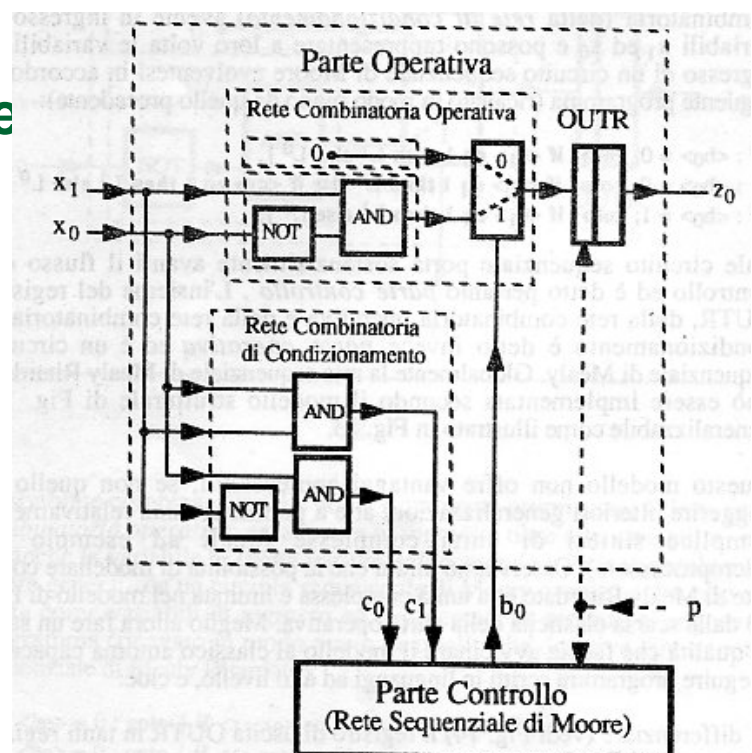
```
L0: <b0>=0; goto {if <c1> eq 1 then L1 else L0}
```

```
L1: <b0>=0; goto {if <c0> eq 1 then L2 else if <c1> eq 1 then L1 else L0}
```

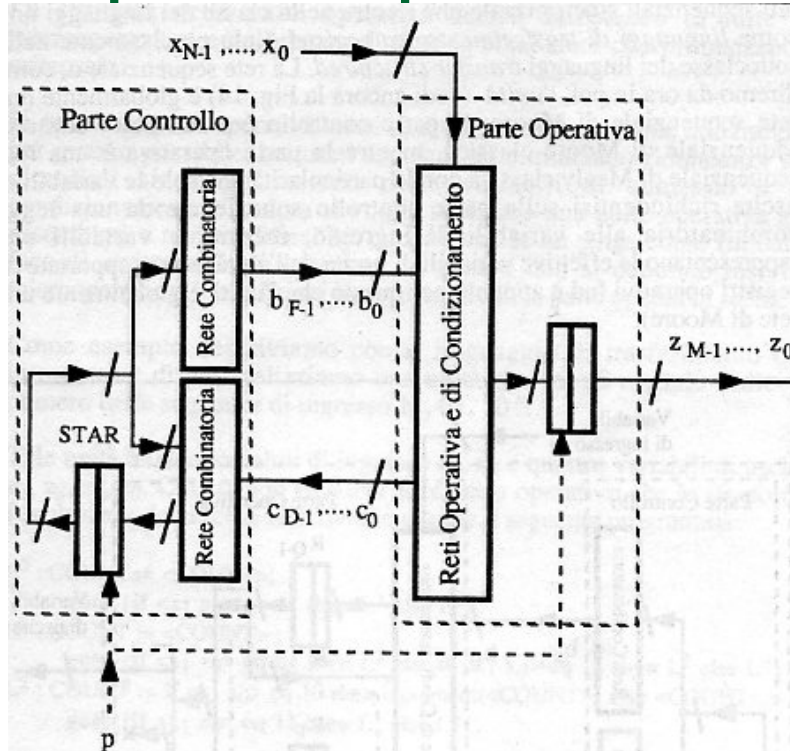
```
L2: <b0>=1; goto {if <c1> eq 1 then L1 else L0}
```

L'insieme del registro OTR, la rete combinatoria operativa e la rete combinatoria di condizionamento è invece detto **parte operativa**.

## Modello di Mealy ritardato per il riconoscitore di sequenza con parte operativa e parte controllo



## Modello generalizzato di Mealy ritardato con parte operativa e parte controllo



## Considerazioni sul modello generalizzato di Mealy Ritardato con PO e PC

- Scarsa elasticità della parte operativa.
- Conviene migliorare il modello in modo da poter eseguire i programmi scritti in linguaggio ad alto livello. Pertanto occorre:
  - differenziare il registro OUTF in tanti registri operativi  $R_0 \dots R_{Q-1}$ ;
  - i registri operativi possono essere utilizzati anche per risultati iniziali o intermedi oltre che per le var. di uscita;
  - I registri operativi possono essere anche accessi in lettura.

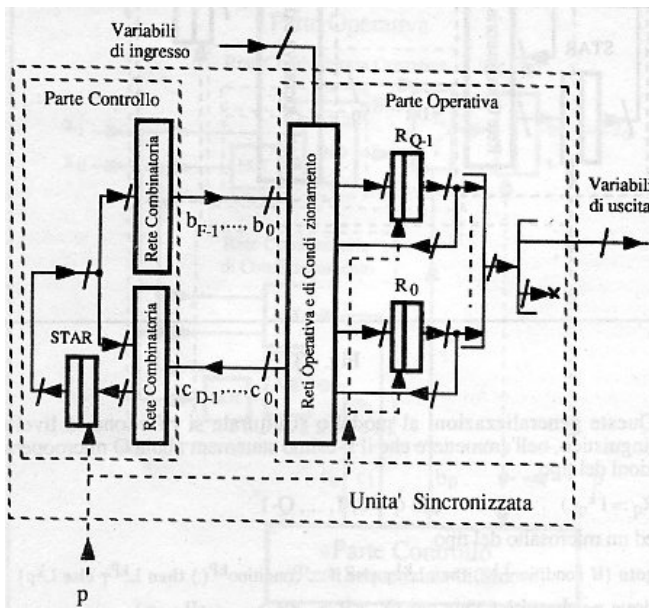
Pertanto ora il k-esimo statement ha:

- $Q$  *microoperazioni* del tipo  $R_q = f^k(\cdot)$  con  $q = 0, \dots, Q-1$
- un microsalto del tipo

```
goto {if conditionk1(.) then Lk1T else if ...
      conditionkP(.) then LkPT else LkF}
```

dove (.) sta per  $(X, \langle R_0 \rangle, \dots, \langle R_{Q-1} \rangle)$ .

# Unità sincronizzata



- PC è RSS di Moore
- PO è RSS di Mealy con
  - var. effettiva di uscita seguite dai registri
  - var. di uscita in ingresso alla PC attraversano una rete combinatoria.
- Globalmente è una RSS di Moore

## Es.: unità con 2 ingressi e 4 uscite che conta modulo 16 il numero delle sequenze 11, 01, 10

```
L0: <COUNT>:=<COUNT>; goto {if <x1 x0> eq 11 then L1
    else L0}
L1: <COUNT>:=<COUNT>; goto {if <x1 x0> eq 01 then L2
    else if <x1 x0> eq 11 then L1 else L0}
L2: <COUNT>:=if <x1 x0> eq 10 then increment{<COUNT>}
    else <COUNT>; goto {if <x1 x0> eq 11 then L1 else L0}
```

- Variabili di condizionamento:

```
c1 = if <x1 x0> eq 11 then 1 else 0
```

```
c0 = if <x1 x0> eq 01 then 1 else 0
```

- Microoperazioni:

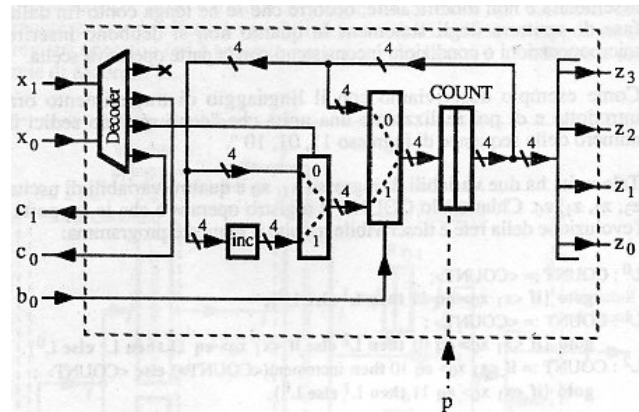
```
COUNT:=case <b0> of
    { 0 : <COUNT>
      1 : if <x1 x0> eq 10 then inc{<COUNT>}
          else <COUNT>}
```

# Linguaggio di trasferimento

## Formato dello statement:

etichetta: microoper., ..., microop.; microsalto

## Esempio: PO e PC



### ■ PO:

- La sottorete `inc` implementa la funzione `<COUNT>`
- Ogni clock se var. `b0 = 0` viene eseguita la prima microoperazione, se `b0 = 1` la seconda.

### ■ PC:

```
L0: <b0>=0; goto {if <c1> eq 1 then L1 else L0}
L1: <b0>=0; goto {if <c0> eq 1 then L2 else if <c1>
    eq 1 then L1 else L0}
L2: <b0>=1; goto {if <c1> eq 1 then L1 else L0}
```

---

## Microsalti

- condizionato a 2 vie:

`goto {if condition(.) then Lu else Lh}`

- incondizionato:

`goto Lu`

---

## Microoperazioni

- Rq = costante
- Rq = input\_data
- Rq = <Rv>
- Rq = increment(<Rv>)
- Rq = decrement(<Rv>)
- Rq = addition(<Rv>)
- Rq = carry(<Rv>)
- Rq = borrow(<Rv>)
- Rq = shift\_left(<Rv>)
- Rq = shift\_right(<Rv>)

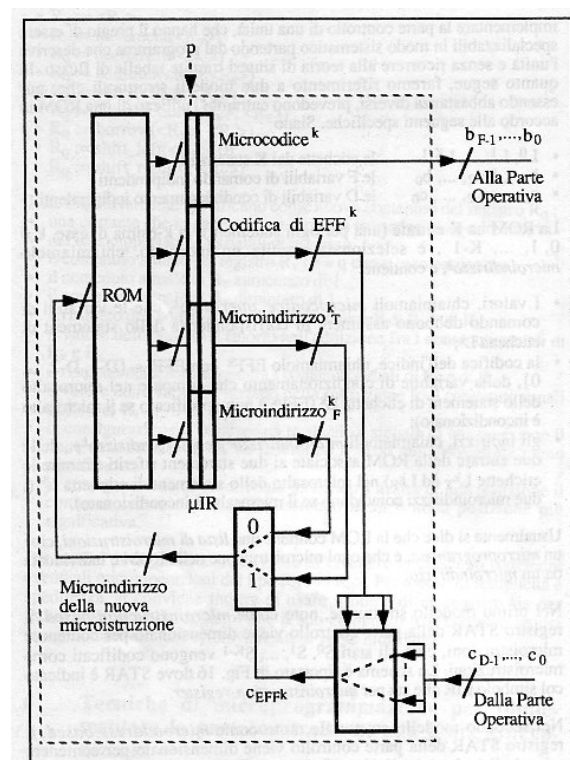


## Tecniche di microprogrammazione per implementare la PC di unità sincronizzate

- Facciamo riferimento a 2 modelli strutturali che prevedono l'utilizzo di una ROM.
- La ROM ha K entrate (da 0 a K-1, una per ogni statement).
- Ogni entrata della ROM è selezionata tramite un microindirizzo<sup>k</sup> e contiene:
  - i valori che le variabili di comando devono assumere in corrispondenza dello statement  $L^k$  – microcodice operativo;
  - la codifica dell'indice  $EFF^k$  della var. di condizionamento che appare nel microsalto di etichetta  $L^k$ ;
  - gli indirizzi: microindirizzo<sup>T</sup> e microindirizzo<sup>F</sup>.
- La ROM contiene una lista di microistruzioni, cioè un microprogramma e ogni microistruzione della ROM è individuata da un microindirizzo.

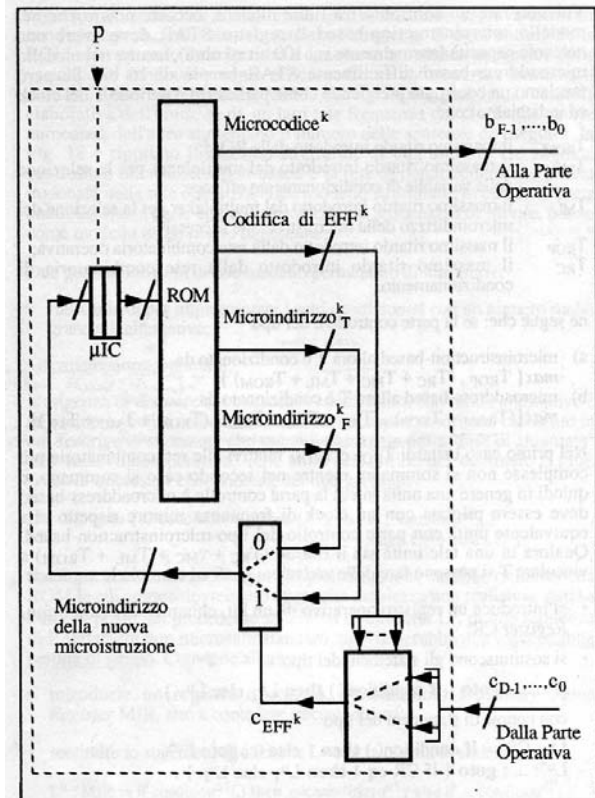
## Modello *microinstruction-based*

- Il registro STAR della parte controllo ( $\mu IR$ ) contiene le microistruzioni, cioè gli stati  $S^0, S^1, \dots, S^{K-1}$ , vengono codificati come microistruzioni.
- Il registro STAR deve essere molto grande (+ di 100 bit).



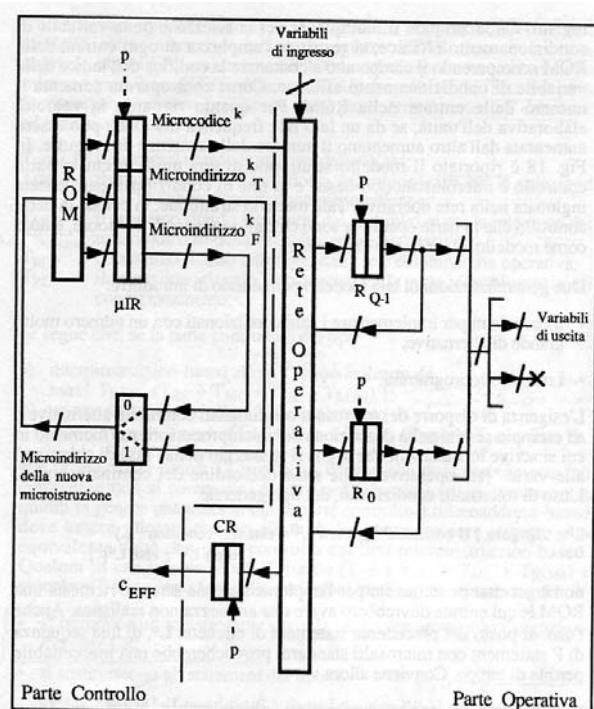
## Modello *microaddress-based*

- Il registro STAR della parte controllo ( $\mu IR$ ) contiene i microindirizzi, cioè gli stati  $S^0, S^1, \dots, S^{K-1}$ , vengono codificati come microindirizzi.
- + Il registro STAR ha un formato ridotto (es. 16 bit)



## Modello ad un *livello di pipeline*

- Se inglobiamo la rete i condizionamento nella parte operativa, utilizzando un registro CR ad 1 bit che tiene conto del condizionamento:
- scompare il multiplexer per la selezione di EFF,
- aumenta la dimensione della ROM.



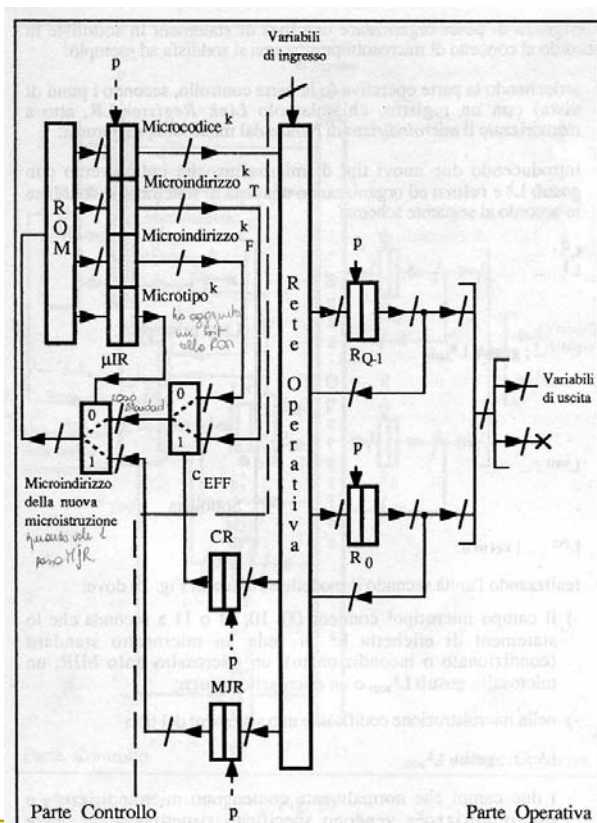
## Microsalti con più alternative (1/2)

- Non possiamo fare statement generali con microsalti a più alternative: la dimensione della ROM aumenterebbe troppo.
- Non è accettabile dal punto di vista dei tempi avere tanti statement con microsalti standard a due vie:
  - Introduciamo un registro operativo: Multiway Jump Register (MJR) atto a contenere microindirizzi.
  - Scriviamo gli statement in questo modo:
 

```

          Lu: MJR:=if conditionu1(.) then
            microindirizzou1T else if ... conditionuP(.)
            then microindirizzouPT else
            microindirizzouPF } ; goto Lu*
          Lu*: ... ; goto MJR.
          
```
  - Quindi realizziamo l'unità secondo il modello illustrato nella figura seguente: (→ slide successiva)

## Microsalti con più alternative (2/2)



- Il microtipo<sup>k</sup> contiene
  - 0, se lo statement di etichetta L<sup>k</sup> prevede un microsalto standard (condizionato o incondizionato),
  - 1, se lo statement di etichetta L<sup>k</sup> prevede il microsalto goto MJR.

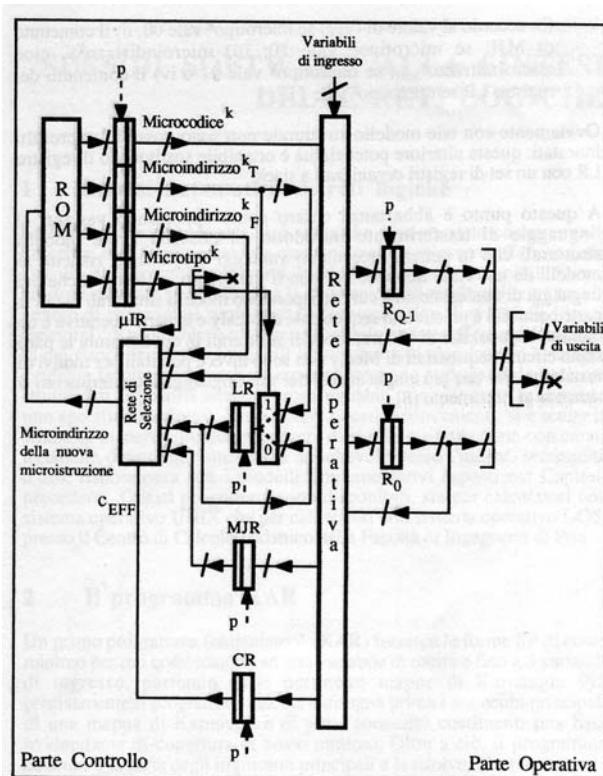
# Microprogrammi

$L^0$ : ...  
 $L^1$ : ...  
 ...  
 $L^u$ : ... ; g o s u b  $L^{sott}$   
 ...  
 $L^{sott}$ : ...  
 ...  
 $L^{ret}$ : ... ; r e t u r n

Occorre:

- aggiungere alla parte operativa un registro chiamata *Link Register* (LR), atto a memorizzare il microindirizzo di rientro dal microsottoprogramma.
- introdurre due nuovi tipi di microalto
  - g o s u b  $L^h$
  - r e t u r n .

# Modello per i microprogrammi



- La rete di selezione sceglie come indirizzo della istruzione successiva:
  - microindirizzo<sup>k<sub>T</sub></sup> o microindirizzo<sup>k<sub>F</sub></sup>, oppure
  - il contenuto di MJR, se il microtipo vale 10, oppure
  - il microindirizzo<sup>k<sub>sott</sub></sup> se il microtipo vale 01, oppure
  - il contenuto del registro LR se il microtipo vale 11.

---

## Considerazioni finali

- Esistono diversi modelli strutturali ognuno caratterizzato da un proprio linguaggio di trasferimento.
- Questi modelli costituiscono l'architettura dei moderni microprocessori, influenzandone le caratteristiche e le potenzialità offerte.
- L'evoluzione temporale di queste unità costituisce l'evoluzione sequenziale degli stati delle CPU.