Microarchitettura: Livello 1

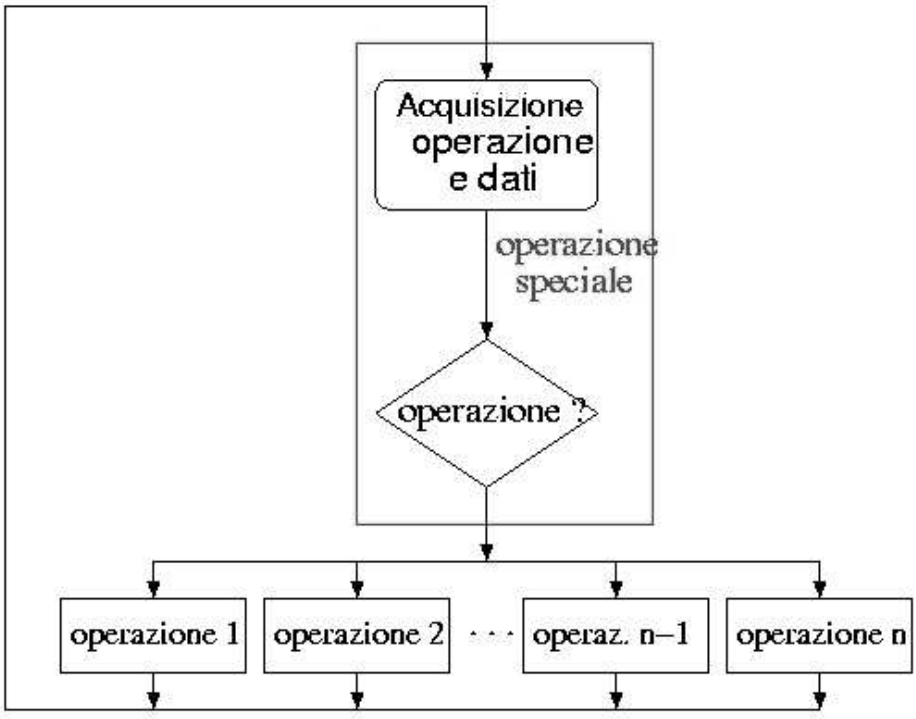
Fino ad ora si è parlato di un elaboratore a livello di circuito digitale. In particolare si è fatto riferimento a circuiti combinatori per la realizzazione di funzioni aritmetico-logiche e di reti sequenziali, che permettono la memorizzazione e la manipolazione con stato della informazione. Ad un livello di astrazione superiore (livello 1), si fa riferimento alla così detta microarchitettura, cioè il sistema di calcolo è visto come composizione di n unità di elaborazione interagenti ed autonome (dati gli ingressi ottenuti tramite interazione con altre unità). Ad ogni unità è affidato un certo sottinsieme delle funzionalità del sistema e l'interazione tra tali unità fornisce la funzione complessiva di sistema. In particolare, un elaboratore generico comprende molte unità (CPU, memoria, I/O, etc.) che contribuiscono a realizzare o supportare vari aspetti del calcolo. Ad esempio, una parte della CPU, detta parte operativa, si preoccuperà di eseguire i calcoli aritmetico-logici, mentre un'altra parte, detta parte controllo, si occuperà di far fluire i dati all'interno delle componenti digitali opportune della CPU affinché la parte operativa possa funzionare correttamente. Inoltre, i risultati parziali e finali del calcolo saranno memorizzati nella memoria principale (RAM) e quindi la CPU leggerà e/o scriverà dati in memoria principale. Se necessario, alcuni dati saranno letti dall'esterno tramite dispositivi di ingresso /uscita (I/O) e così via.

Il livello 1 è anche detto livello firmware. Il livello firmware è interpretato direttamente (eseguito) dall'hardware (livello 0: circuiti digitali) mediante una opportuna interconnessione di reti combinatorie e sequenziali.

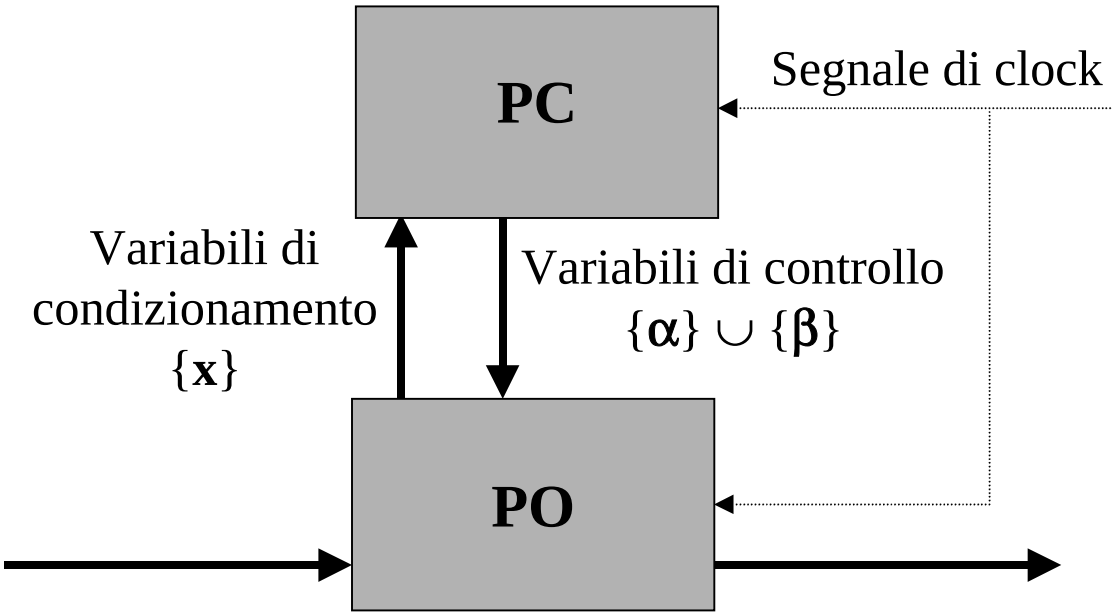
Ciascuna unità a livello 1 opera in modo sequenziale, con funzionamento specificato da un *microprogramma (μprogramma)* espresso in un dato *micro-linguaggio (μlinguaggio)* che permette di programmare la interazione fra le reti combinatorie e sequenziali del livello 0. In particolare, ogni unità a livello firmware è in grado di eseguire sequenze di operazioni. Le operazioni che formano tali sequenze, dette operazioni esterne dell' unità, sono indipendenti fra loro ed appartengono ad un insieme predefinito di operazioni che sono realizzabili dalla unità considerata. Una operazione esterna di una unità deve essere intesa come un servizio che l'unità mette a disposizione di altre unità.

L'esecuzione di ogni diversa operazione esterna è descritta da un diverso frammento di *μprogramma* (detto interprete dell'operazione). Esiste anche una operazione speciale che di fatto realizza l'acquisizione dall'esterno dei dati e del codice (nel senso di identificatore di operazione, e non di "codice macchina eseguibile") della operazione esterna da eseguire.

Il *μprogramma* di una unità riunisce i frammenti di programma delle diverse operazioni (esterne e speciale). Il *μprogramma* ha una struttura ciclica in cui si alterna l'esecuzione della operazione speciale con l'esecuzione della operazione esterna il cui codice e dati da eleborare sono stati acquisiti dalla operazione speciale (vedi figura sotto).



L'interpretazione del microprogramma viene usualmente eseguita da due reti sequenziali LLC interagenti, denominate Parte Controllo (PC) e Parte Operativa (PO). La PC costituisce l'automa di controllo dell'unità. Essa ha tanti stati interni quante sono le μistruzioni del μprogramma (lo stato corrente determina la μistruzione in esecuzione). La PO esegue le operazioni elementari previste da ogni μistruzione quindi dispone di: registri, reti combinatorie (ALU, …), circuiti per l'instradamento dei dati (multiplexer, bus, …).



La PO provvede all'esecuzione dei comandi del microlinguaggio (microistruzioni) tramite reti combinatorie standard e registri. La PC provvede al:

- controllo di sequenzializzazione delle microistruzioni tramite variabili di condizionamento {x} relative allo stato interno di PO;
- invio comandi di esecuzione a PO tramite variabili di controllo {α} ∪ {β};

Le variabili di condizionamento {x} di fatto costituiscono una funzione {x}: PO → PC. Un esempio di variabile di condizionamento è il risultato di un test per zero del contenuto di un certo registro (che ovviamente appartiene alla PO).

Le variabili di controllo, invece, costituiscono una funzione {α} ∪ {β}: PC → PO, dove di fatto:

- i segnali {β} abilitano / disabilitano la scrittura nei registri di PO;
- i segnali {α} forniscono gli ingressi secondari per i commutatori, selezionatori, ALU, etc. di PO.

Ad esempio, le variabili di controllo possono determinare l' instradamento da registro/i sorgente a registro destinazione della microistruzione.

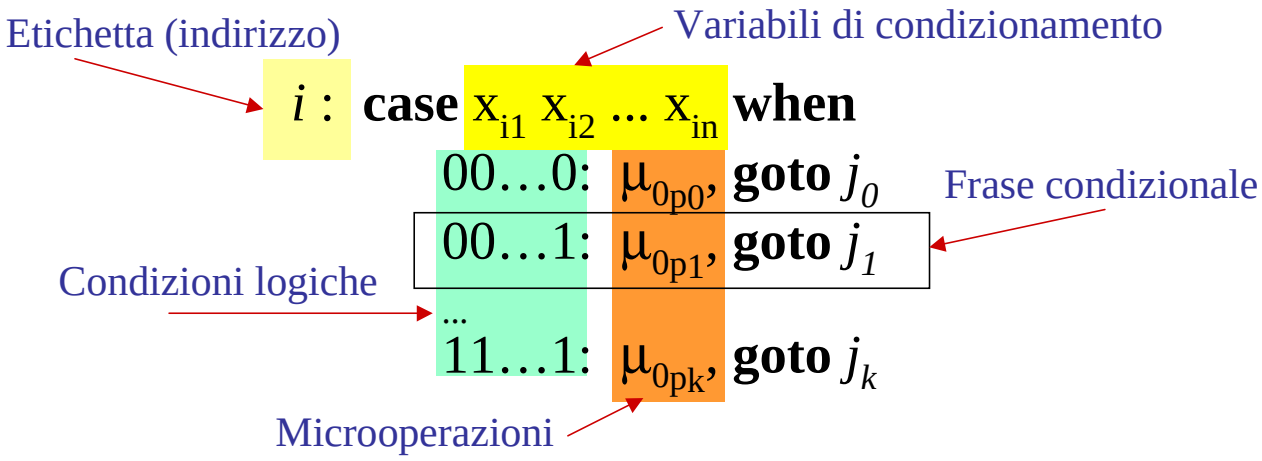
Per meglio comprendere il concetto di microprogrammazione di una unità, vediamo quali sono i passi da seguire per progettare una unità microprogrammata.
Di seguito riportiamo in sequenza la lista ordinata delle attività che bisogna svolgere:

1. specifica delle operazioni esterne dell'unità;
   in questa fase bisogna definire quali sono le operazioni esterne, cioè i servizi, che l'unità mette a disposizione dell'ambiente in cui si trova, cioè a disposizione di altre unità che possono accedere ai servizi della unità che stiamo progettando;
2. scrittura del microprogramma di interpretazione delle operazioni esterne;
   una volta definite le operazioni esterne, bisogna scrivere la sequenza di μistruzioni (vedremo meglio in seguito cosa esse sono) che di fatto implementano ognuna delle operazioni esterne; ciò viene fatto assumendo la disponibilità di variabili che poi dovranno necessariamente essere mappate sull'hardware che costituirà sia la PO che la PC;
3. derivazione di PO a partire dal microprogramma;
   a partire dalla specifica del microprogramma si determina di quali risorse hardware la PO ho bisogno per permettere l'esecuzione del microprogramma e si definiscono di conseguenza i segnali di

condizionamento da inviare alla PC; in particolare, a seconda del livello di prestazioni che si vuole ottenere e del costo che si vuole sostenere, si provvederà a fornire la PO di più o meno registri e/o reti combinatorie per la esecuzione veloce ed efficiente di operazioni aritmetico-logiche;

4. derivazione di PC a partire dal microprogramma;
   come nel caso precedente, bisogna determinare quali particolari segnali di controllo bisogna generare verso la PO, anche considerando i segnali di condizionamento provenienti dalla PO;
5. determinazione del periodo di clock;
   una volta stabilite in che modo sono fatte sia la PO che la PC, si può determinare il periodo di clock più breve che permette la stabilizzazione delle eventuali alee di commutazione delle parti combinatorie sia di PO che di PC;
6. valutazione delle prestazioni.
   si valuta sperimentalmente se la PO e PC lavorano efficientemente insieme; in caso di inefficienze, si cerca di capire come ottimizzare la PO, la PC e le interazioni fra di loro.

Il microlinguaggio utilizzato per definire un microprogramma non è unico. Esistono alcuni linguaggi diversi che si caratterizzano per definizione sintattica diversa delle microoperazioni e per la conseguente realizzazione della PO e PC. Ad esempio, nel Microlinguaggio Phrase Structured (PS) una microistruzione è definita nel seguente modo:

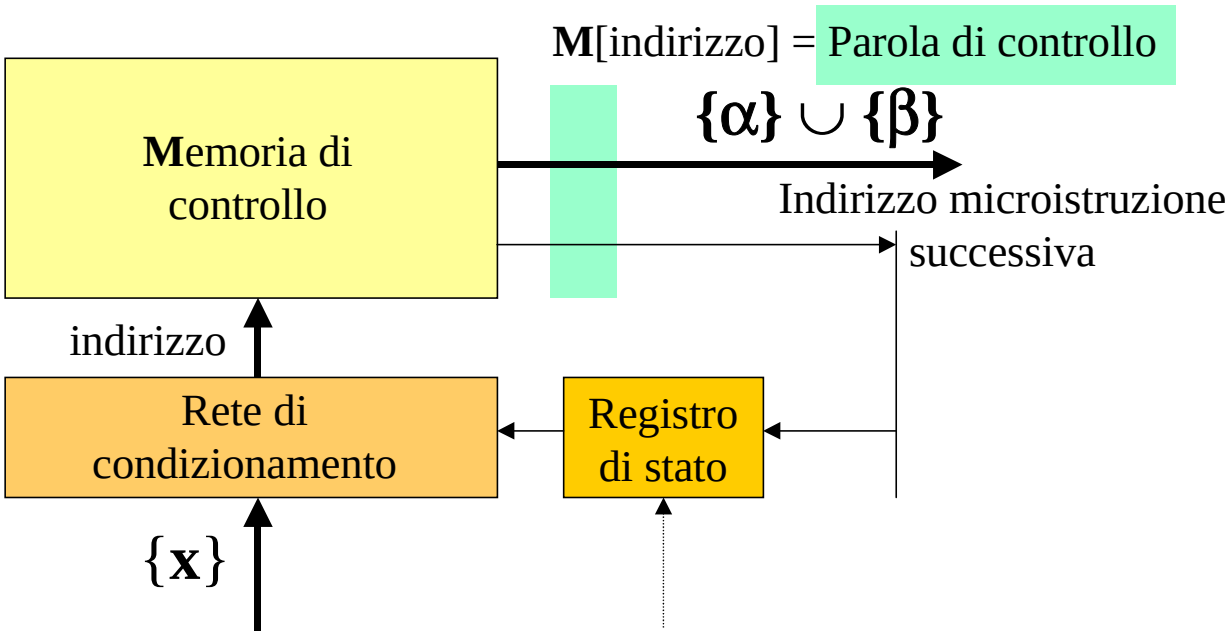


Ogni microistruzione del microprogramma:

- è caratterizzata da una etichetta (indirizzo);
- ha una condizione sulle variabili di condizionamento (condizione di case);
- a seconda delle condizioni logiche soddisfatte al momento della esecuzione (configurazione binaria delle variabili di condizionamento) esegue una particolare microoperazione (che consiste nella emissione di una particolare stringa di segnali $\{\alpha\} \cup \{\beta\}$;
- conseguentemente alla emissione dei segnali di controllo, l'esecuzione salta alla microistruzione la cui etichetta è specificata dopo la microoperazione eseguita.

Si noti che ogni microistruzione viene eseguita in un ciclo di clock, e che una microoperazione può essere costituita da una operazione nulla (*nop*) o da trasferimenti, anche multipli, tra registri.

Per quanto riguarda la realizzazione di PO e PC tramite reti sequenziali, la PO è tipicamente vista come rete sequenziale di Moore e se si utilizzano microoperazioni non parallele basta una ALU multi-funzione per realizzare le operazioni aritmetico-logiche necessarie per la elaborazione. La PC è invece molto più frequentemente realizzata secondo il modello di rete sequenziale di Mealy. In particolare, si può comprendere la PC come un automa a stati dove ad ogni etichetta di microistruzione corrisponde uno stato interno dell'automa, alle possibili configurazioni delle variabili di condizionamento corrispondono stati di ingresso, mentre le microoperazioni (che costituiscono le variabili di controllo) rappresentano gli stati di uscita.
Mentre la PO deve necessariamente essere realizzata in hardware (registri, addizionatori, ALU, etc.) la parte controllo può essere realizzata direttamente in circuiti hardware, o in alternativa può essere realizzata con una tecnica di microprogrammazione. In quest'ultimo caso si parla di PC microprogrammata (vedi figura sotto):



Nel caso di PC microprogrammata le possibili configurazioni di variabili di controllo (in sostanza l'insieme delle microoperazioni che costituiscono l'intero microprogramma di una unità) vengono memorizzate in una Memoria di Controllo, una microoperazione per cella di memoria. La microoperazione da eseguire ad un dato istante di tempo è selezionata grazie alla associazione dello stato corrente (contenuto in un registro di stato impulsato dal clock) e delle variabili di condizionamento correnti con l'indirizzo in Memoria di Controllo che contiene la microoperazione da eseguire. Questa associazione avviene tramite una Rete di Condizionamento. Nella Memoria di Controllo, oltre ai segnali di controllo ($\{\alpha\} \cup \{\beta\}$), sono memorizzati gli stati successivi della PC. Quindi dalla Memoria di Controllo in effetti si legge una parola di controllo costituita in parte dalle variabili di controllo da generare e per la restante parte dalla codifica dello stato successivo in cui l'automa deve entrare (al successivo impulso di clock). Tale stato successivo sarà poi utilizzato in congiunzione con le successive variabili di condizionamento per selezionare all'interno della Memoria di Controllo la parola di controllo succesiva. Quindi, si può capire subito che con questo approccio non è molto costoso cambiare un microprogramma in quanto essenzialmente basta cambiare in modo opportuno il contenuto della Memoria di Controllo, senza bisogno di intervenire a livello dell'hardware. Ovviamente, il prezzo che si deve pagare per tale flessibilità è un tempo di esecuzione più lento della PC.

Oltre alla realizzazione della PC come modello di Mealy e alla simultanea realizzazione della PO come modello di Moore, esistono altre possibilità realizzative. Nel caso in cui sia la PO che la PC siano realizzate tramite modello di Moore, il linguaggio di microprogrammazione assume una sintassi particolare basata essenzialmente su una struttura a trasferimento. Per questo motivo si parla di microlinguaggio Transfer-Structured (TS). Di seguito è riportata la struttura generale di una microistruzione nel microlinguaggio TS:

```
etichetta : microoperazione
        case condizione logica
                when valore 1 => indirizzo successivo 1
                when valore 2 => indirizzo successivo 2
                …
                when valore n => indirizzo successivo n
        end case
```

Come conseguenza della organizzazione Moore-Moore, un microprogramma viene eseguito in un maggior numero di cicli di clock, che tuttavia sono di minor durata rispetto al modello Mealy-Moore. Esiste anche la possibilità di utilizzare una organizzazione Moore-Mealy che tuttavia è equivalente al modello Moore-Moore, con PO ottenuta *anticipando* il prelievo di {x} all'*ingresso* dei registri.

Di seguito riportiamo un esempio di specifica di una operazione esterna e come questa possa essere realizzata da microprogrammi in microlinguaggio PS (che assume una organizzazione Mealy per la PC e Moore per PO) e microlinguaggio TS (sia nel caso di una organizzazione Moore-Moore e Moore-Mealy).

Specifica:
while true **loop**
A := A + B;
if A < 0 **then** A := -A **end if**;
end loop;

Formalismi di μprogramma:
; : azioni *alternative*
, : azioni *parallele*
Le condizioni logiche si *omettono* se *tutte* abilitano la *stessa* azione

PS (Mealy-Moore): [*ottimizzato*]
0. A + B → A, 1
1. (A_0 = 1) –A → A, 0 ; (A_0 = 0) A + B → A, 1

TS (Moore-Moore):
0. A + B → A, 1
1. *nop* (A_0 = 0) 0 ; (A_0 = 1) 2
2. -A → A, 0

TS (Moore-Mealy):
0. A + B → A (A_0 = 0) 0 ; (A_0 = 1) 1
1. -A → A, 0

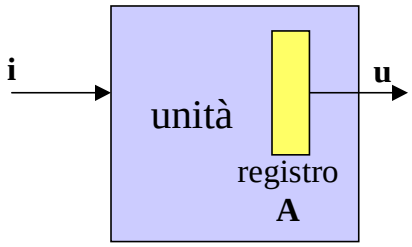
[{x} *anticipato*]

Per meglio comprendere i concetti espressi in precedenza, di seguito vediamo come procedere per realizzare la PO e PC di una unità molto semplice microprogrammata.
In particolare, faremo le seguenti assunzioni semplificative:

- supponiamo che l'unità da realizzare implementi una sola operazione esterna (vedi specifica sotto) che richiede un ingresso **i** (numero rappresentato in complemento a 2);
- per semplificare non si considera l'operazione speciale, cioè si assume che:
  - l'unità riceva l'ingresso **i** quando l'unità è pronta per l'elaborazione;
  - l'uscita **u** della unità (memorizzata internamente in un registro **A**) è prelevata da chi ne ha bisogno prima che questa venga sovrascritta dalla uscita successiva

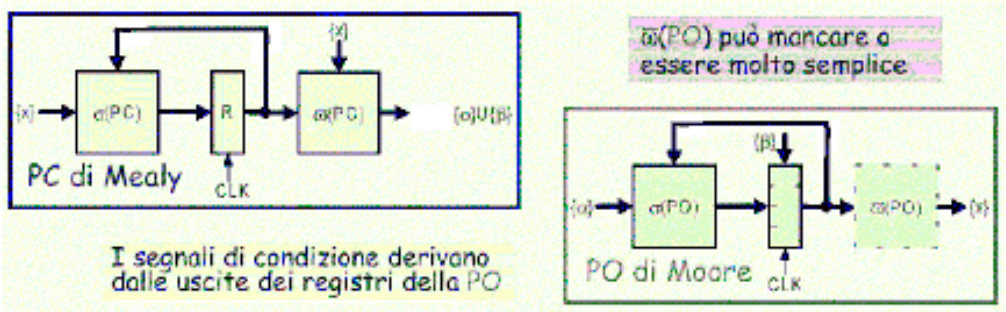
Inoltre i considera una organizzazione Mealy-Moore e quindi un microlinguaggio PS. Di seguito è riportata la specifica della operazione esterna e la sua realizzazione tramite un microprogramma (non ottimizzato) in microlinguaggio PS.

Specifica:
while true **loop**
A := A + i; /* i ingresso */
if A < 0 **then** A := -A **end if**;
end loop;

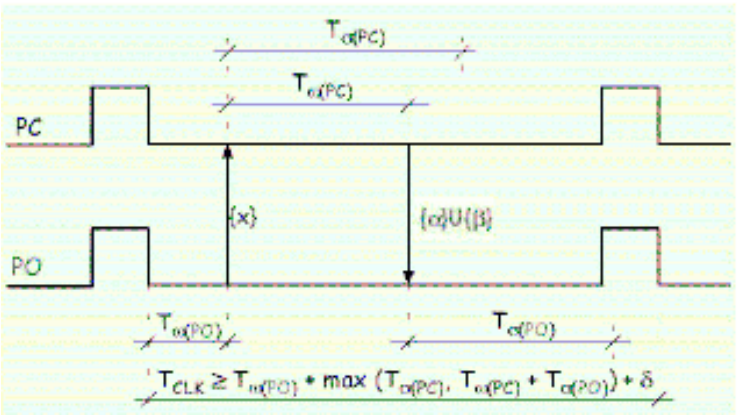


PS (Mealy-Moore): [*non ottimizzato*]
0. A + i → A, 1
1. (A_0 = 1) –A → A, 0 ; (A_0 = 0) nop, 0

Prima di procedere con la specifica della PO e quindi della PC a partire del microprogramma, è bene dare alcuni nozioni aggiuntive su come la PO interagisce con la PC. Di seguito è mostrata lo schema realizzativo della PC come automa di Mealy e lo schema realizzativo della PO come automa di Moore. Il funzionamento di questi due automi è completamente specificato dalle reti combinatorie che costituiscono le funzioni di transizione dello stato e le funzioni di uscita degli automi, così come visto quando abbiamo parlato di reti sequenziali. In questo caso, si farà riferimento ad una modalità di interazione seriale (cioè in sequenza) fra la PO e la PC. In questo corso non abbiamo tempo di esplorare altre modalità di interazione. Dalla rappresentazione grafica è possibile accertarsi che sul fronte di discesa del clock, la rete combinatoria che realizza la funzione di uscita della PO genera i segnali di condizionamento che raggiungono la PC. Da tale momento, le reti combinatorie che realizzano sia la funzione di transizione dello stato che la funzione di uscita della PC iniziano a modificare il loro stato a causa delle nuove variabili di condizionamento. In genere il tempo di stabilizzazione della rete combinatoria che realizza la funzione di transizione dello stato è maggiore rispetto al tempo impiegato dalla rete che calcola l'uscita. In ogni caso, appena l'uscita di PC è generata, le corrispondenti variabili di controllo va ad influenzare la reti combinatoria che implementa la funzione di transizione di stato della PO. Quindi l'arrivo dell'impulso di clock determina la sincronizzazione degli stati e quindi segue la generazione del nuovo (e successivo) output per la PO (ricordarsi che la PO segue un modello di Moore) e si riprende con il ciclo visto in precedenza.



- Funzionamento seriale



- T_{CLK} è il periodo di clock
- $T_{\sigma(PC)}$: ritardo massimo della rete combinatoria σ(PC)
- $T_{\sigma(PO)}$: ritardo massimo della rete combinatoria σ(PO)
- $T_{\omega(PC)}$: ritardo massimo della rete combinatoria ω(PC)
- $T_{\omega(PO)}$: ritardo massimo della rete combinatoria ω(PO)
- δ ritardo prop.+ setup flip-flop

Andando a calcolare i vari ritardi di "assestamento" delle reti coinvolte sia nella PC che nella PO, non è difficile calcolare il periodo minimo di clock che garantisce il corretto funzionamento del sistema (vedere disequazione in basso a sinistra della figura di sopra).

A questo punto, chiarito come la PO interagisce con la PC e viceversa, possiamo tornare al nostro esempio realizzativo di unità microprogrammata.
In particolare, possiamo osservare che la PO ha bisogno di:

- una ALU in grado di effettuare sia l'addizione (α=0) che la negazione (per il cambio di segno; α=1);
  - in ingresso la ALU ha **i** (ingresso alla unità) e il contenuto del registro A, ed α è il segnale di controllo che seleziona l'operazione da eseguire;
- un registro A, la cui scrittura è abilitata con segnale β=1 e disabilitata con segnale β=0;
  - il segno del valore contenuto in A è stabilito dal bit A_0 del registro A (A_0 costituisce il segnale di condizionamento x).

Inoltre lo stato interno di PO è costituito dal contenuto del registro A (se il registro è a n bit, allora la PO avrà 2^n stati interni), e la funzione di uscita di PO (modello Moore) è l'identita (infatti l'uscita u coincide con il contenuto del registro A, cioè lo stato).
La PC è una rete sequenziale di Mealy dotata di 1 ingresso binario (A_0), 2 uscite binarie (α, β) e 2 stati interni ("0", "1"). In particolare, la PC:

- nello stato "0", qualunque sia il valore assunto da A_0, la PC deve generare i segnali α=0 (A+i → A) e β=1 (registro A abilitato);
- nello stato "1", se A_0=1, la PC deve generare i segnali α =1 (-A → A) e β=1 (registro A abilitato);
- nello stato "1", se A_0=0, la PC deve generare i segnali α =* (non importa il valore di α) e β=0 (registro A disabilitato).

Il tutto si può riassumere nella seguente tabella di verità:

Stato \ Ingresso	A_0=0	A_0=1
"0"	"1" $\alpha\beta$=01	"1" $\alpha\beta$=01
"1"	"0" $\alpha\beta$=*0	"0" $\alpha\beta$=11

Per la corrispondente realizzazione hardware della PO e PC si rimanda ai lucidi della lezione 6.
Ovviamente, in alternativa ad una realizzazione hardware per la PC, si poteva adottare l'approccio microprogrammato che utilizza una memoria di controllo discusso in precedenza.