Modelli strutturali reti sequenziali

Il transitorio nelle reti

- Una rete combinatoria ideale è definita dal mapping Y=f(X) in cui X e Y sono vettori di variabili booleane e il tempo di calcolo della rete è nullo, la rete commuta istantanemante
- Una rete combinatoria fisica, realizzata mediante circuiti elettronici, presenta per effetto della limitatezza della propagazione dei segnali (al massimo $3*10^8$ m/sec) e degli effetti capacitivi e induttivi parassiti dei circuiti, effetti di deformazione dei segnali e di ritardo che seppur piccoli (dipendenti dalla tecnologia, per CMOS ad es. ritardo dell'ordine del nsec o frazione di esso) sono significativi e creano fenomeni che genericamente vanno sotto il nome di alee

Ritardi nelle reti combinatorie

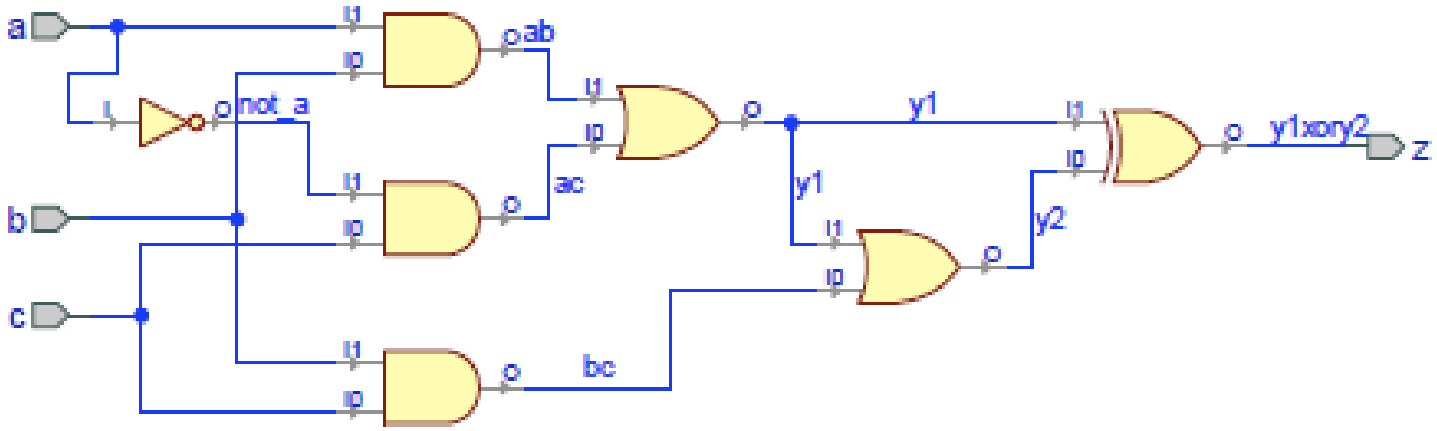
- Tralasciando gli aspetti della deformazione dei segnali, i ritardi propri dei circuiti logici con cui si propagano i segnali dipendono dai percorsi che i segnali effettuano durante la loro propagazione nel circuito.
- I tempi di propagazione sono caratterizzabili mediante valori minimi e massimi
- Attendendo un tempo sufficientemente lungo affinché si possa esaurire il transitorio dovuto alla propagazione dei segnali interessati, la rete combinatoria presenta all'uscita il valore previsto dalla sua tabella di specifica.
- Durante il transitorio l'uscita della rete può assumere valori non sempre controllabili e coincidenti con lo stato finale in cui deve portarsi. Tali valori possono generare fenomeni non desiderati nelle reti collegate alla rete combinatoria ad essi interessata, alee nel funzionamento

Il fenomeno delle alee

- Le alee si possono classificare in alee statiche e alee dinamiche
- Alee statiche rete and-or
- Alee statiche rete or-and
- Verifica in tempi diversi sulle uscite di transizioni provocate dal cambiamento dello stato di ingresso
- Formazione di uno o più impulsi spuri sulle uscite a causa di alee

Sincronizzazione nelle reti

- Consentire la consultazione delle uscite solo alla fine dei transitori
- Sincronizzare il funzionamento di un utilizzatore con quello della rete
- L'utilizzatore deve conoscere l'istante o l'intervallo di tempo in cui i segnali possono essere interrogati e, quindi, deve avere informazioni sul funzionamento delle reti a monte o sui segnali provenienti dai circuiti con cui esso è connesso



Questo esempio mostra la generazione di un'alea statica in una rete combinatoria and-or a più livelli. L'alea statica è generata al 1° livello di tale rete a causa di differenti ritardi nella

Segnali a livelli e segnali a impulsi

Elementi di memoria

- Linee di ritardo
- Flip-flop
- Celle ROM e similari

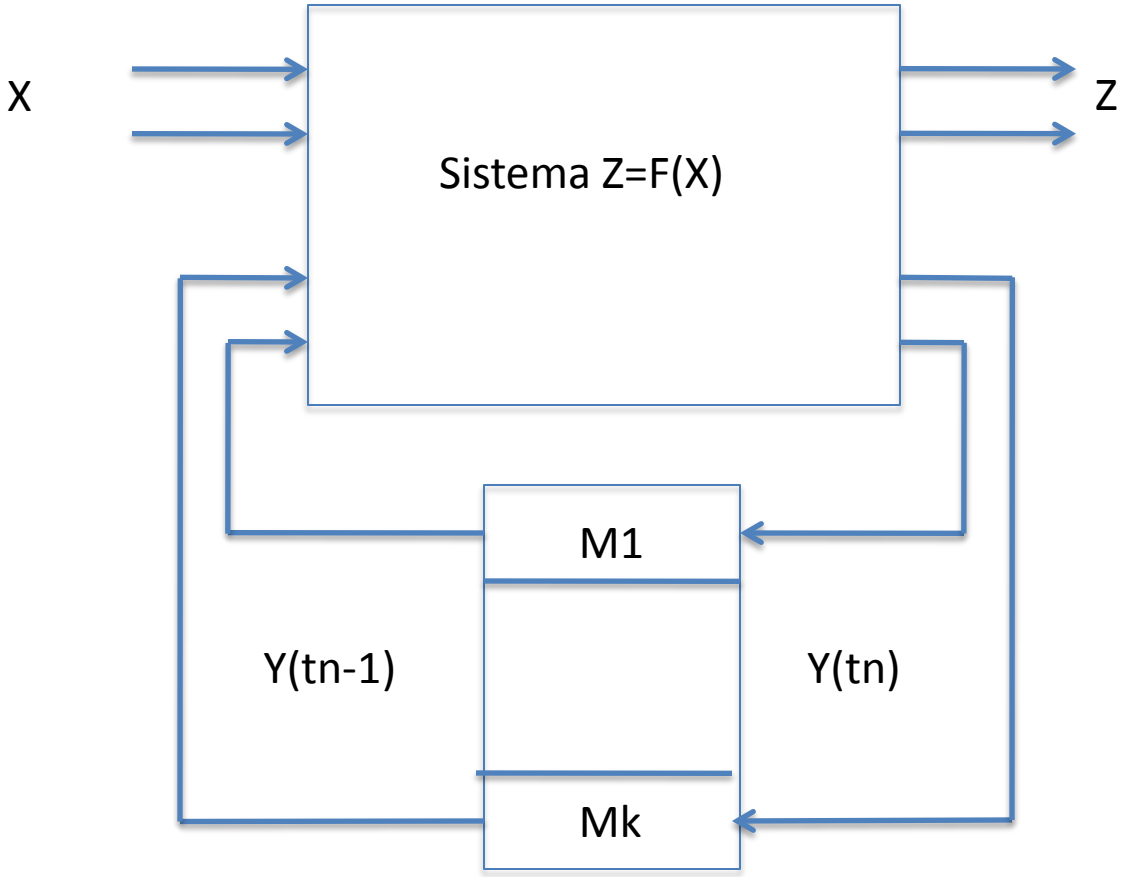
Flip flop

- Latch
- Edge triggered
- Clocked e non
- Master-slave

Dalla macchina (FSM) alla rete sequenziale

- FSM
  - è un modello astratto basato su simboli, stati e funzioni algebriche
  - Concetto di macchine equivalenti e compatibili, completamente e incompletamente specificate
  - Macchine minime
- Rete sequenziale
  - Variabili rappresentate da segnali
  - Uso di porte logiche dotate di ritardo e comportamento non ideale e che generano alee
  - Linee di interconnessione non ideali ma con ritardo e deformazione dei segnali

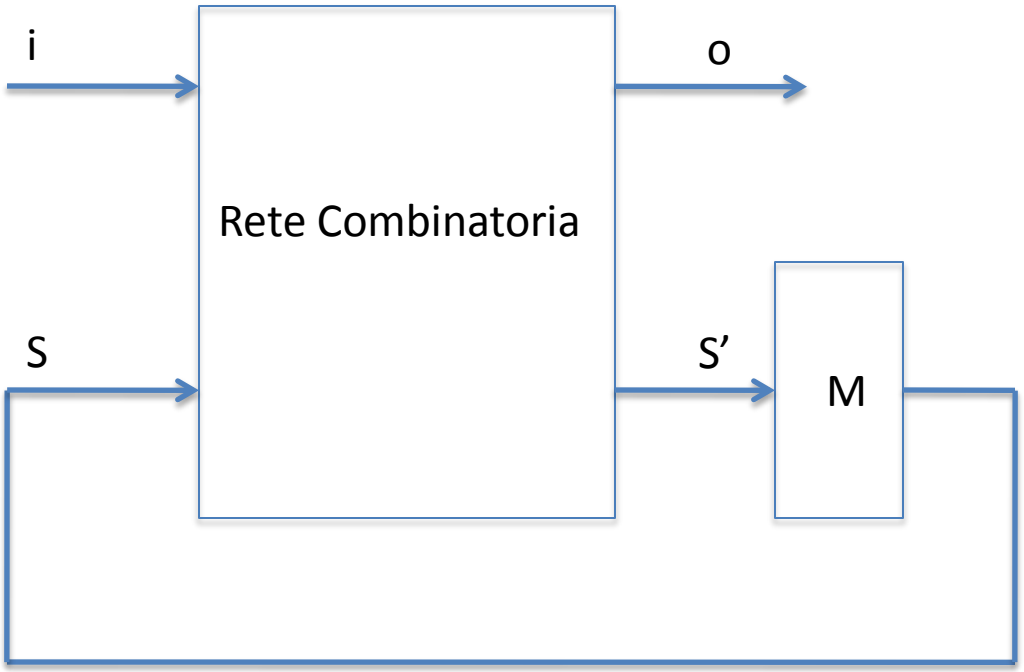
Sistema sequenziale



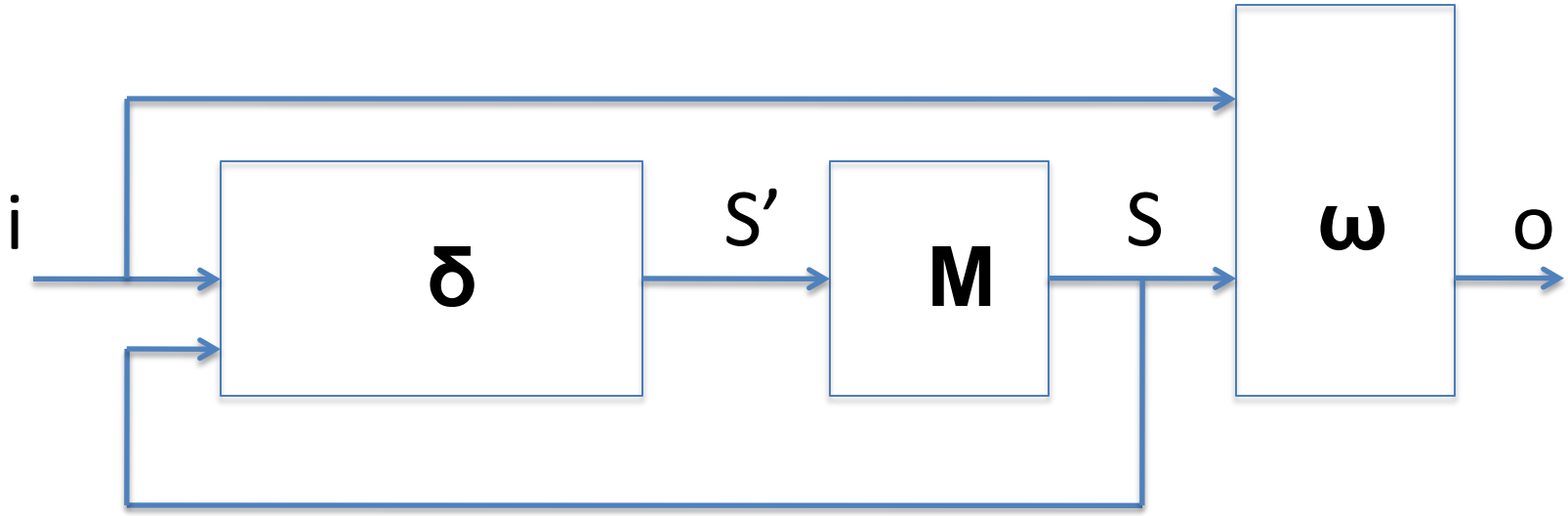
Codifica dei segnali e alfabeto dei simboli

- Invece di considerare segnali binari codificati con sequenze 01, si opera per studiare il modello del sistema sequenziale sui simboli di ingresso, uscita e stato (presente e prossimo).
- Con n segnali binari si codificano 2^n simboli per cui si opera con alfabeti I, O, S aventi rispettivamente cardinalità 2^j, 2^h e 2^k.

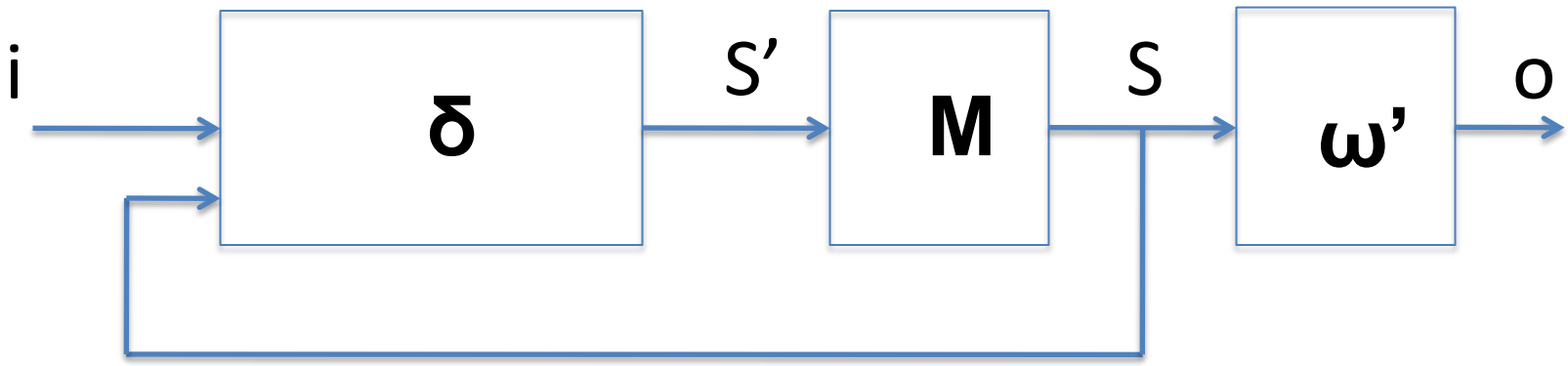
Modello strutturale ideale



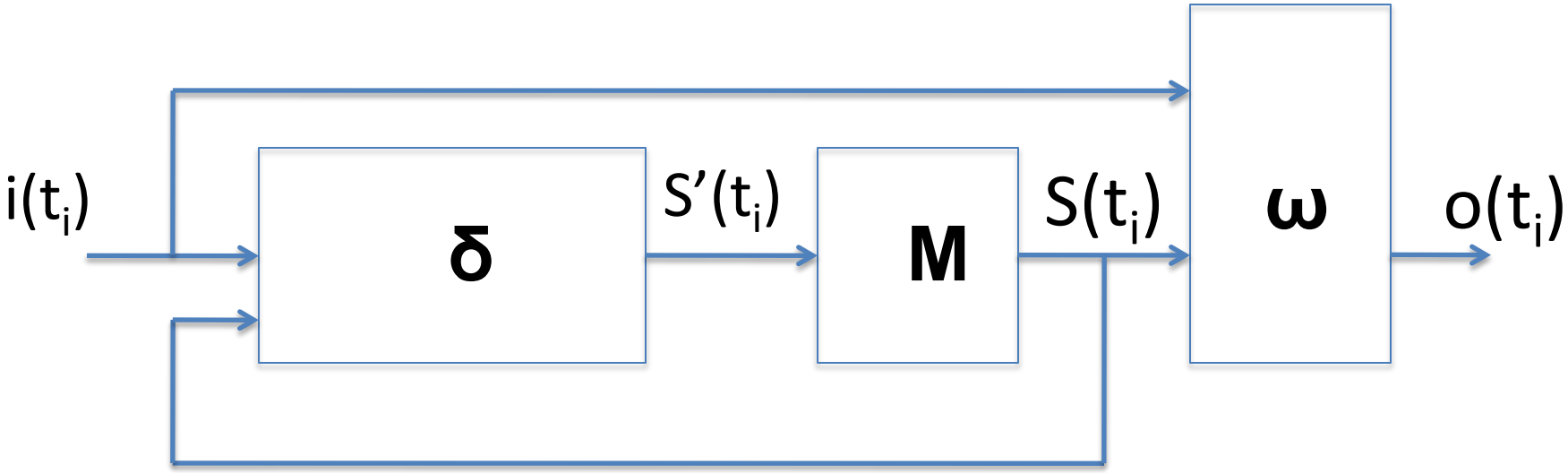
Schema di macchina di Mealy



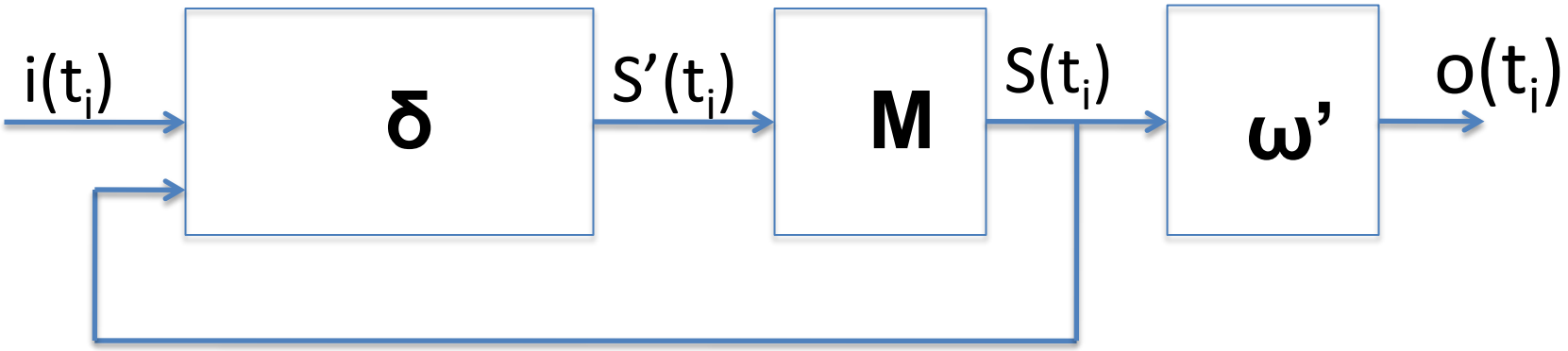
Schema di macchina di Moore



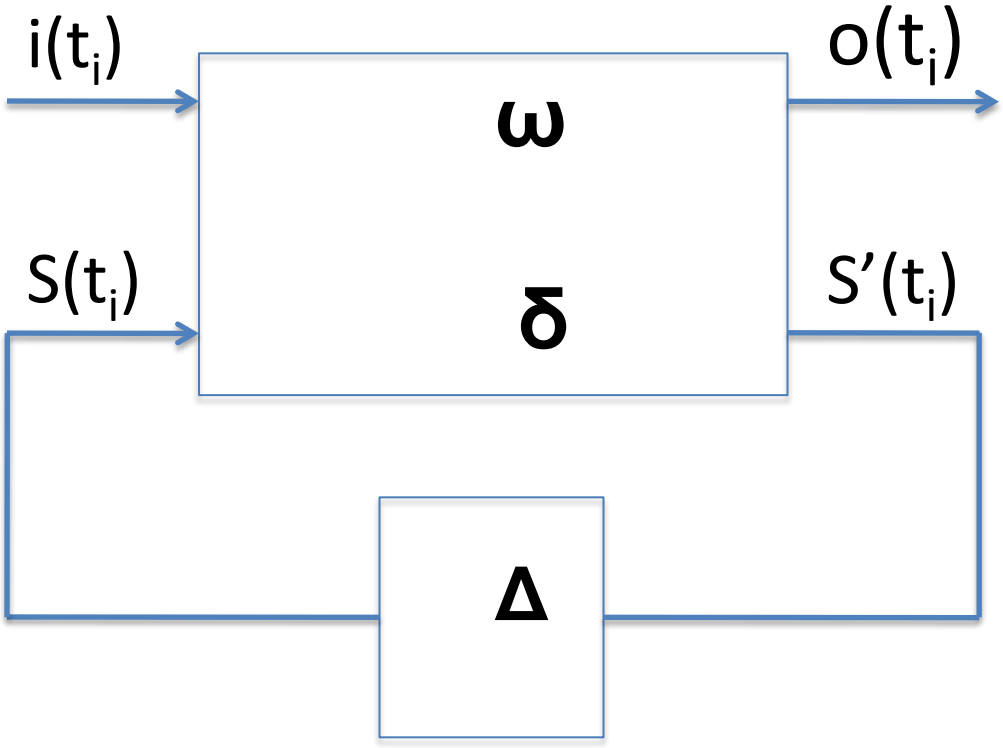
Modello strutturale ideale per macchina di Mealy



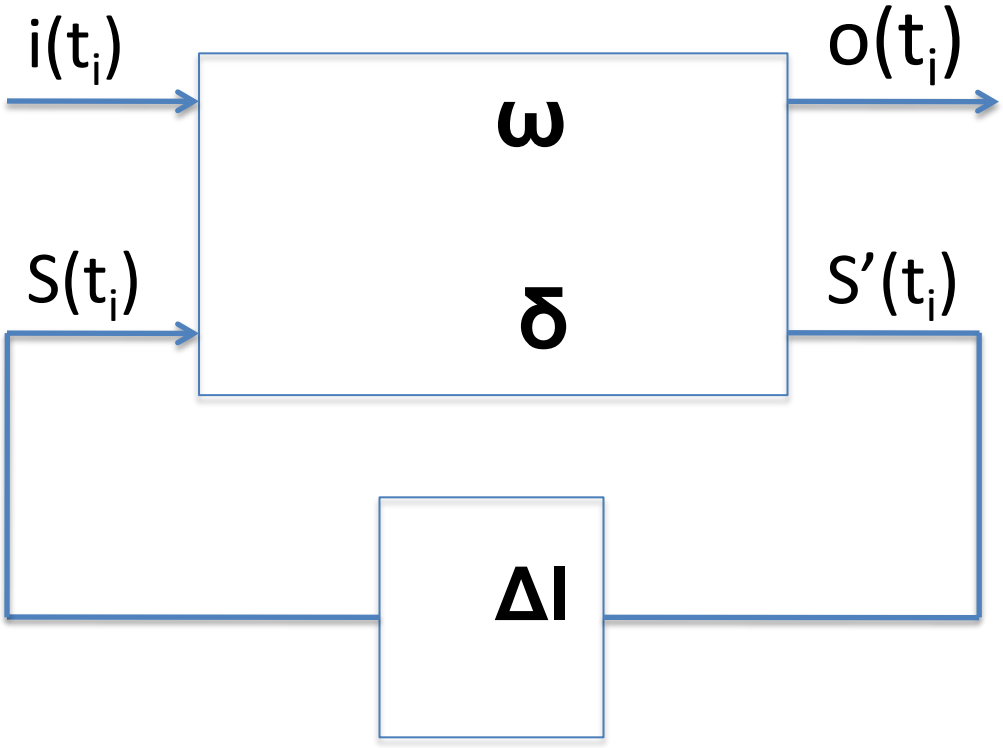
Modello strutturale ideale per macchina di Moore



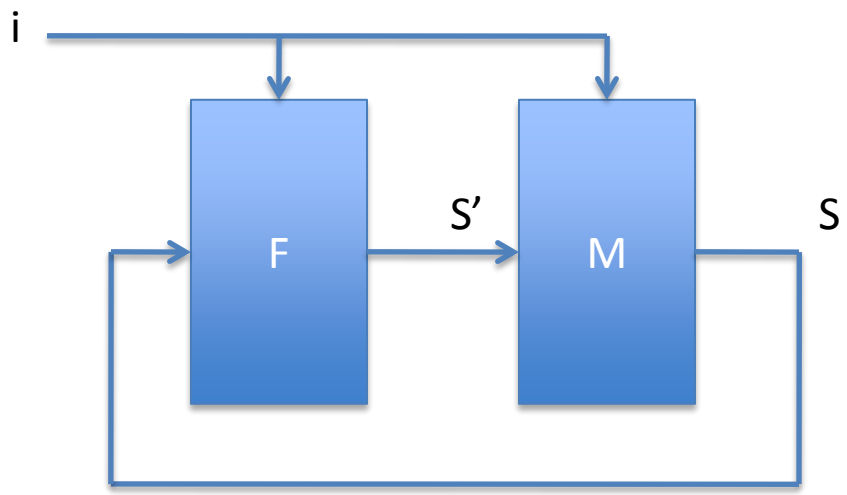
Modello strutturale ideale



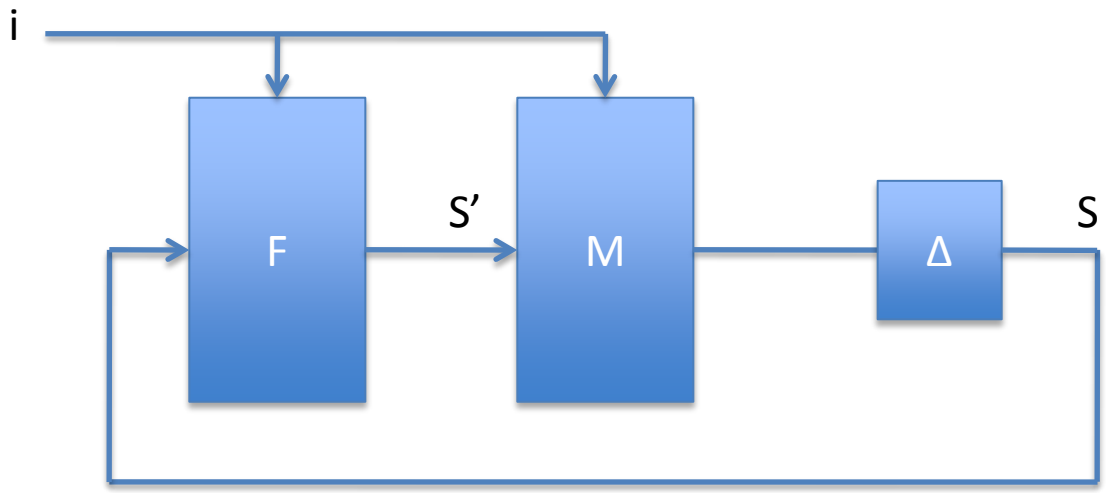
Modello strutturale per reti impulsive



Modello strutturale di macchina impulsiva



Modello strutturale di macchina impulsiva con Δ



Definizioni

- Definition 1. Stato totale
  - Stato totale è la coppia ingresso-stato <i,s>
- Definition 2. Stato totale stabile
  - Stato totale stabile è uno stato totale tale che δ(i,s)=s
- Definition 3. Stato stabile
  - Uno stato s si dice stabile se e solo se \forall<i,t δ(i,t)=s\Rightarrow\delta(i,s)=s
- Definition 4. Stato instabile
  - Uno stato si dice instabile se non è stabile

Definizioni

- Definition 5. Macchina asincrona
  - Una macchina si dice asincrona se e solo se, partendo da uno stato stabile, cambiando l'ingresso, in un numero finito di passi, si sposta in un altro stato totale stabile.
- Definition 6.
  - Macchina asincrona SOC una macchina asincrona è detta SOC (Single Output Change) se, partendo da uno stato totale stabile, cambiando l'ingresso si osserva al più una variazione del simbolo d'uscita.

Definizioni

- Definition 7. Macchina asincrona MOC
  - Una Macchina asincrona è detta MOC (Multiple Output Change) se, partendo da uno stato totale stabile, cambiando l'ingresso si osserva più di una variazione del simbolo di uscita.
  - Le macchine asincrone di tipo MOC sono di scarso interesse pratico e non saranno studiate.
- Definition 8. Macchina asincrona SOC normale
  - una macchina asincrona SOC si dice normale se è caratterizzata da stati interni tutti stabili
- Definition 9. Macchina sincrona
  - Una macchina è sincrona se non è asincrona

Trasformazione di una macchina sequenziale in una macchina impulsiva

- Definizione 10: data una macchina di Mealy $\langle I,S,O,\delta,\omega \rangle$, la macchina impulsiva corrispondente è la quintupla $\langle I_p,S,O_p,\delta_p,\omega_p \rangle$ tale che:
  - $I_p = I \cup \{i_0\}$;
  - $O_p = O \cup \{o_0\}$;
  - $\delta_p(i,s) = \delta(i,s)$; $\delta_p(i_0,s) = s$

Macchina asincrona vs sincrona

- Una macchina asincrona mette in corrispondenza univoca la sequenza di uscita con quella di ingresso, indipendentemente dalla durata dei simboli di ingresso (sotto il vincolo fisico che la durata di questi deve almeno essere uguale al ritardo Delta introdotto per la memorizzazione dello stato).
- Ciò non è vero per una macchina sincrona in cui il funzionamento è corretto solo se si vincola il cambiamento di un simbolo di ingresso con il cambiamento dello stato, cioè l'istante in cui lo stato successivo diventa lo stato presente coincide con l'istante in cui cambia il simbolo di ingresso

Anomalie macchine sincrone impulsivi

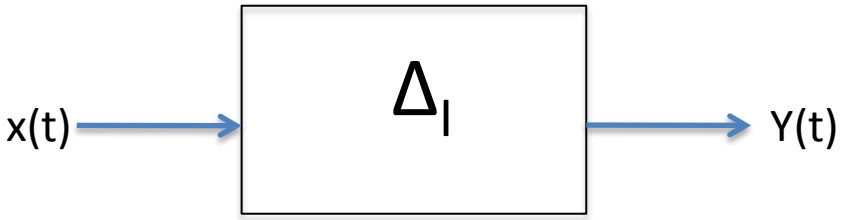
- Le anomalie di funzionamento delle macchine sincrone impulsive sono dovute al fatto che i simboli di ingresso, persistendo per un periodo maggiore di Δ , determinano un comportamento non univoco della macchina stessa.
- Per evitare le transazioni multiple si può imporre di vincolare la durata di un simbolo di ingresso in un intervallo W tale che sia verificata la relazione $\Delta \leq W < 2\Delta$; ciò implica anche il filtraggio delle transazioni spurie causate da simboli di ingresso di durata $W < \Delta$ mediante l'inserimento di un ritardo inerziale Δ_I

Ritardo Inerziale ΔI

$$(1a) \quad Y(\tau)\ non\ definito\ \forall \tau \in [0,\Delta)$$

$$(1b) \quad Y(t+\Delta) = x(t)\ iff\ \ x(t+\varepsilon) = x(t) \forall \varepsilon \in (0,\Delta]$$

$$altrimenti\ Y(t+\Delta) = Y(t)$$



- Un ritardo inerziale è un elemento di ritardo Δ la cui uscita y(t) replica il nuovo ingresso x, variato all'istante t, se e solo se esso perdura più di Δ, altrimenti mantiene il valore precedente:
  - la condizione (1°) evidenzia la condizione di Y(t) definita solo a partire da Δ;
  - La condizione (b) esprime la condizione a regime del ritardo inerziale così come precedentemente definito

Definizione di macchina di sequenziale

- Il modello di Mealy è definito dalla quintupla:
- $MS_{Mealy}=<\mathbf{I,S,O},\delta,\omega>$ con
- $I=\{i_1,i_2,\ldots i_m\}$ alfabeto di ingresso
- $S=\{s_1,s_2,\ldots s_m\}$ degli stati interni
- $O=\{o_1,o_2,\ldots o_m\}$ di uscita
- $\delta=IxS\rightarrow S$ funzione stato succ.
- $\omega=IxS\rightarrow O$ funzione uscita

Definizione di macchina di sequenziale

- Il modello di Moore è definito dalla quintupla:
- MS_{Moore}=<**I,S,O**,δ,ω'> con
- I={i_1,i_2,…i_m} alfabeto di ingresso
- S={s_1,s_2,…s_m} degli stati interni
- O={o1,o2,…om} di uscita
- δ=IxS→S funzione stato succ.
- ω'=S→O funzione uscita

Tre tipi di rappresentazione

- Diagramma degli stati
  - un grafo orientato e etichettato caratterizzato da n nodi e m-n rami orientati colleganti coppie di nodi;
  - ogni nodo rappresenta uno stato, ogni ramo è etichettato da una coppia i/o
- Tabella stati/uscite
  - Ha n=|S| righe e m=|I| colonne, l'elemento generico a_{hk} della tabella è dato da $<\delta(i_k s_k),\omega(i_k,s_k)>$
  - Per Moore la tabella presenta una colonna in più che contiene le uscite $<\omega(s_k)>$ e l'elemento generico a_{hk} contiene $<\delta(i_k s_k)>$

Tre tipi di rappresentazione

- Matrice di connessione
  - Ha n righe (stati presenti) e n colonne (stati successivi), tante quanto sono gli stati, l'elemento generico a_{ij} è costituito da tutte le coppie <i,o> tali che $\delta(i,s_i)=s_j$ e $\omega(i,s_i)=o$

Sequenze di ingresso-stato-uscite

- I sistemi combinatori effettuano trasformazioni dei simboli di un alfabeto in simboli di un altro alfabeto
- I sistemi sequenziali effettuano trasformazioni di sequenze o successioni di simboli di un alfabeto in sequenze di simboli di un altro alfabeto

Funzioni su sequenze

- Sia **I** un alfabeto di m simboli, I* è definito come l'insieme di tutte le sequenze generabili con i simboli di I, compresa la sequenza nulla che indichiamo con λ
  - x(yz)=(xy)z per ogni x,y,z in I*
  - $X\lambda=\lambda x=x$
- Indichiamo con J una sequenza di I* e con L(J)=Lj=t la sua lunghezza, si ha
  - $L(J_1+J_2)=L(J_{1)} + L(J_2)$

Rete sequenziale

- Una rete logica in cui i valori i valori assunti in un certo intervallo di tempo non dipendono solo dai valori delle variabili di ingresso nello stesso intervallo di tempo, ma dalla sequenza temporale dei valori delle variabili di ingresso
- L'uscita delle reti sequenziali dipende dalla storia delle configurazioni delle variabili di ingresso
- Completa analogia tra macchina a stati finiti e rete sequenziale

Progettazione rete sequenziale

- Problema di sintesi: associare variabili di commutazione a simboli astratti
- Realizzare una rete sequenziale corrispondente ad una data macchina sequenziale senza alcun vincolo sulla scelta delle variabili di ingresso e di uscita (in quanto la macchina vive in isolamento)
- Progettare una rete logica da inserire in un sistema complesso, cosa che vincola sia le variabili di ingresso che di uscita)
- Meccanismi per la realizzazione fisica della memoria dello stato (soluzioni differenti per asincrone e sincrone)

Progettazione rete sequenziale/2

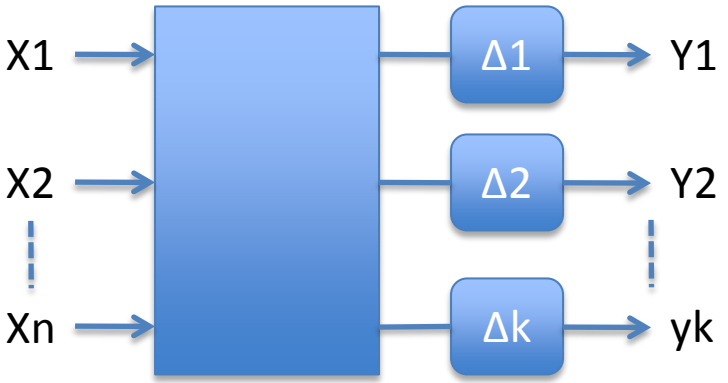
- Individuare la macchina sequenziale che risolve un assegnato problema specificato come FSM
  - Associazione variabili di ingresso e alfabeto I e tra variabili di uscita e alfabeto O
  - Individuare il numero di stati necessari, una funzione δ e una ω che specificano la macchina
  - Minimizzazione della macchina
  - Trasformazione macchina in rete fisica
    - Codifica stati interni con variabili di stato
    - Sintesi della funzioni δ e ω

Realizzazione rete sequenziale implica

- Sintesi delle funzioni di transizione di stato δ e di uscita ω tramite reti combinatorie
- Modello strutturale reti sequenziali che supportano macchine asincrone
- Modello strutturale reti sequenziali che supportano macchine impulsive di tipo sincrono e impulsive asincrone

Tempificazione rete combinatoria

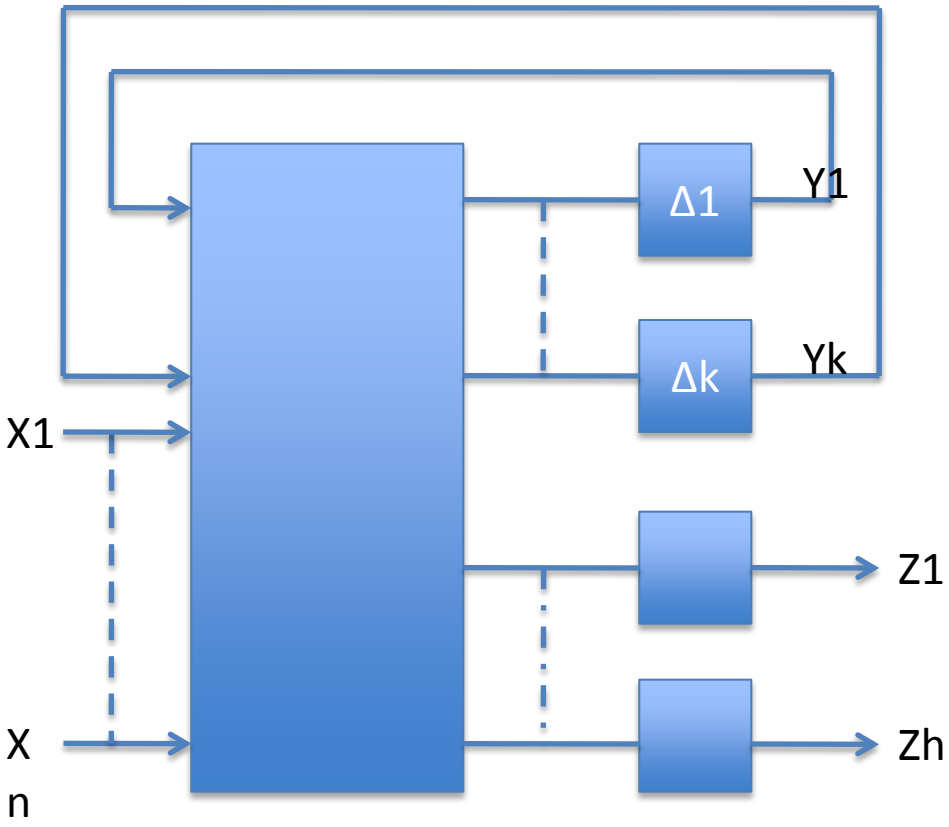
- A seguito della variazione di una variabile di ingresso al tempo t_0 la rete combinatoria produce un'uscita in un tempo che per ciascuna delle variabili di uscita Yi può variare tra un minimo ($t_0+\Delta_m$ e un massimo $t_0+\Delta_M$, con Δ_m ritardo minimo e Δ_M ritardo massimo, ritardi dovuti a differenti percorsi di propagazione dei segnali



Modello strutturale di principio delle reti asincrone

- Reti asincrone in modo fondamentale
  - Le variabili di ingresso possono commutare esclusivamente una alla volta
  - Il modello strutturale non pone alcun vincolo sul tipo di variabili di ingresso, purché rispettino le condizioni del modo fondamentale; possono con esso realizzarsi sia macchine asincrone che macchine asincrone impulsive
  - Il funzionamento in modo fondamentale impone che le sequenze ammissibili sono solo quelle costituite da configurazioni tali che due qualsiasi configurazioni consecutive differiscono per il valore di una sola variabile
  - Nel caso di reti asincrone impulsive, le condizioni impone, inoltre, che per due configurazioni consecutive, una delle due deve essere la codifica della base impulsiva i_0.

Modello strutturale di principio di reti sequenziali che supportano macchine asincrone



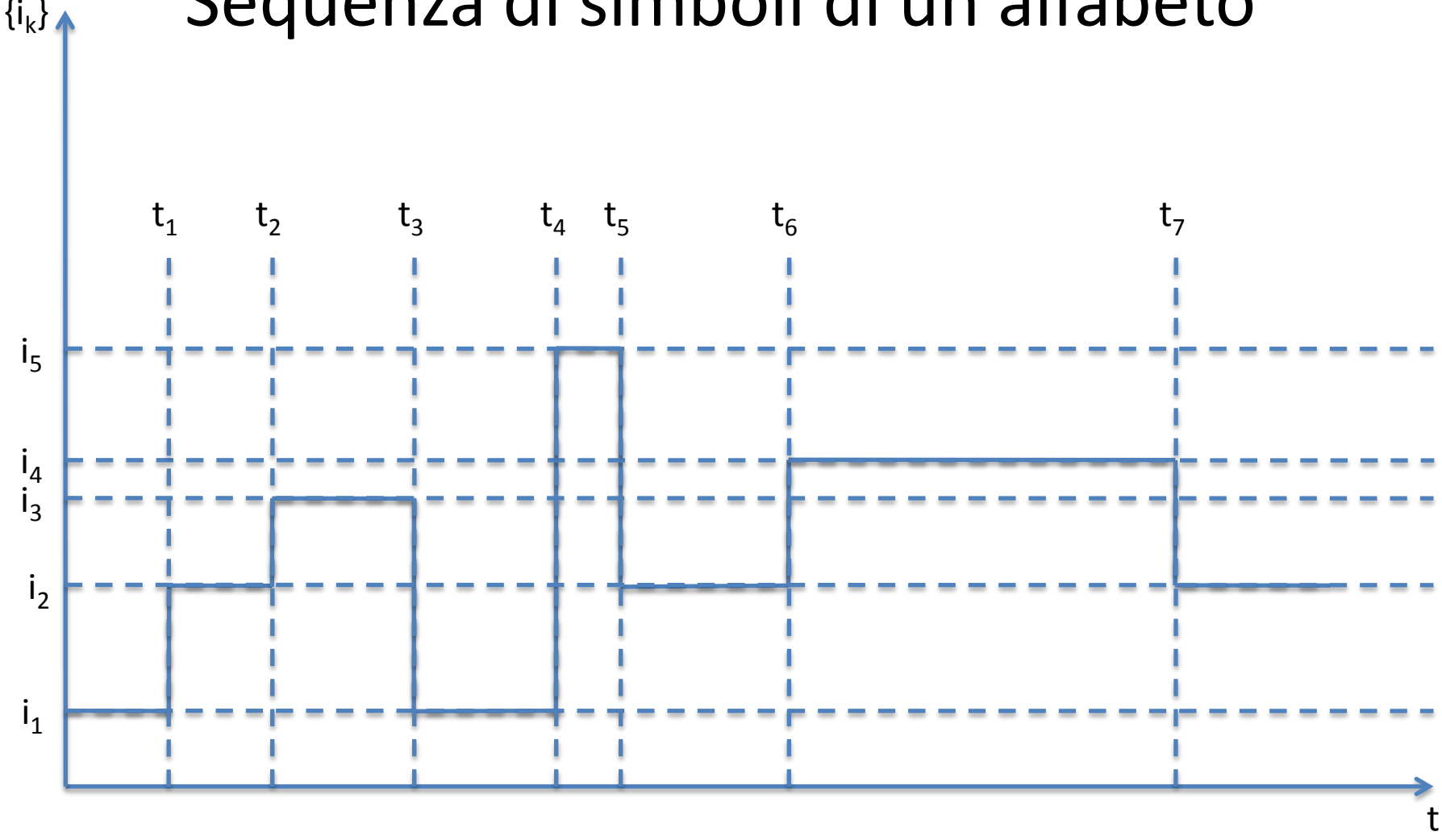
Sequenze

- Con riferimento ad un alfabeto di simboli associati ai valori di una sequenza, detto I lo spazio degli ingressi, si aggiunga a I un particolare simbolo, che definiamo “simbolo spazio” o “valore neutro” i_0 , per cui l'alfabeto è dato da $I=\{i_0,i_1,i_2,...,i_k\}$.
- E' allora possibile definire sequenze di ingresso caratterizzate dall'alternanza simbolo-spazio che chiameremo sequenze impulsive $...i_i\ i_0\ i_j...$
- i_0 non è un evento significativo per la sequenza ma un sostegno per l'impulso che, invece, caratterizza l'evento
- Secondo tale definizione, trattando sequenze logiche e non fisiche (si fa riferimento ad un alfabeto di simboli astratto)
- un impulso può durare un tempo qualunque, è solo vincolante fra un impulso e il successivo il passaggio per lo stato neutro i_0 }.

Uscite per macchine di Mealy/Moore per ingressi impulsivi

- Se si opera secondo il modello di Mealy la macchina con ingressi impulsivi vincola le uscite ad essere anch'esse impulsive
- Se si opera secondo il modello di Moore le uscite, dipendendo soltanto dallo stato, presentano una sequenza non impulsiva
- Una qualunque macchina sequenziale può essere trasformata in impulsiva

Sequenza di simboli di un alfabeto



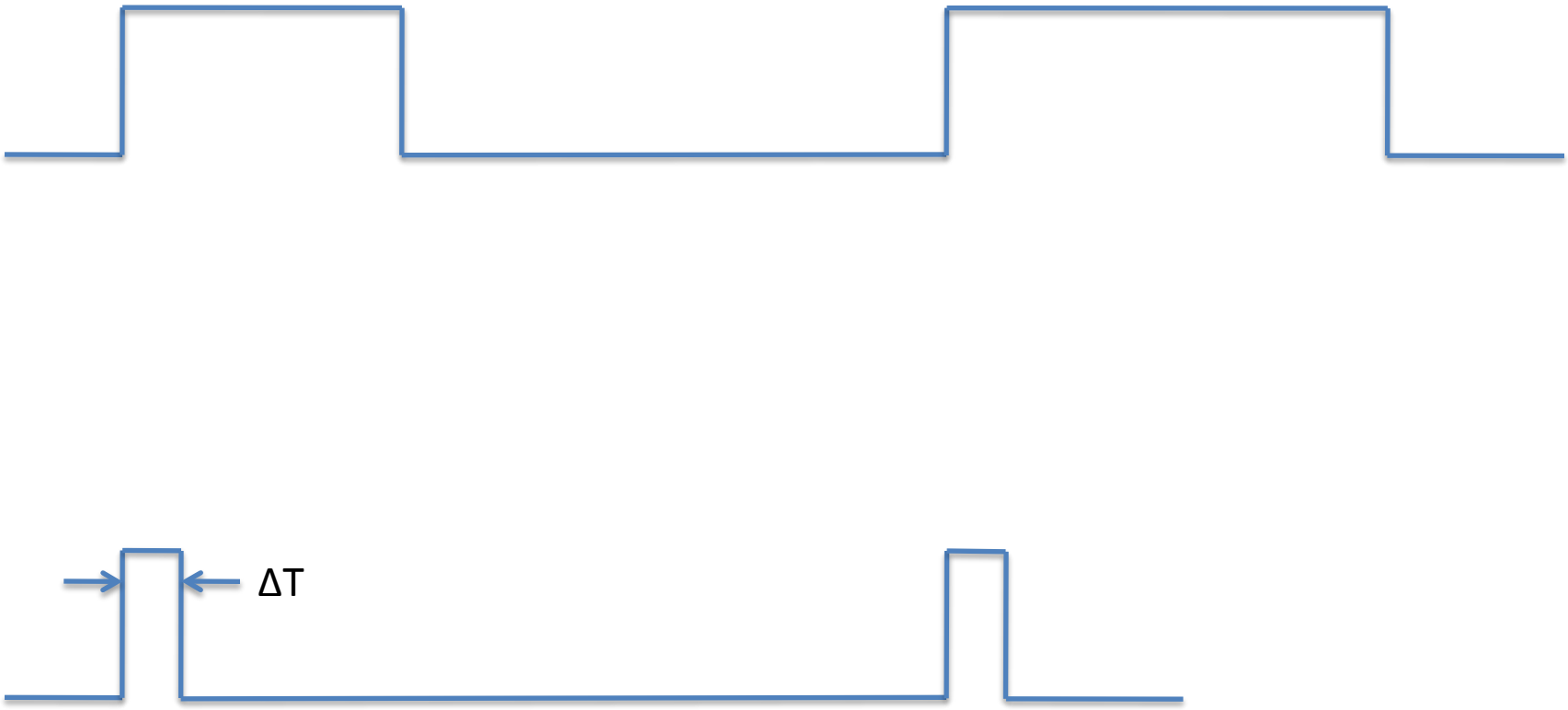
Sequenza: {i1, i2, i3, i1, i5, i2, i4, i2,
…}
Sequenza tempificata: {<t0, i1><t1, i2> <t2, i3> <t3, i1> <t4, i5> <t5, i2 <t6, i4> <t7, i2>…}

Sequenza sincronizzata dall'esterno

Sequenza tempificata esterno: {<e1, i1><e2, i3> <e3, i1> <e4, i2> <e5, i4> <e6, i4> <e7, i2>…}

Codifica alfabeto

- Essendo l'alfabeto di interesse I={i_0, i_1, i_2, i_3, i_4, i_5, i_6, i_7} composto da 8 simboli, questi possono essere codificati mediante 3 segnali binari $b_2b_1b_0$


ΔT

Sequenze di ingresso

- L'alfabeto di ingresso è caratterizzato dalla presenza di n simboli che possono essere usati per generare sequenze di ingresso di varia lunghezza (in teoria infinite sequenze essendo ammesse le ripetizioni di sottosequenze). In ogni sequenza di simboli, ciascun simbolo può essere codificato con ricorso a variabili booelane. La sequenza di simboli deve essere letta dalla macchina mediante un opportuno meccanismo in grado di recepirla.

Sequenze

- Teoricamente è la sequenza che ha senso e che determina il comportamento della macchina e non la durata dei suoi simboli o il modo per codificarli. Per una macchina fisica, però, occorre specificare come i vari simboli si presentano e possono essere letti in base alla loro tipologia. La macchina, per interpretare i simboli, ha bisogno di rilevarne la presenza e il significato (valore). La presenza può essere rilevata solo a causa di una variazione degli ingressi o di un apposito evento in grado di segnalare che il simbolo è presente e può essere letto.

Codifica dei simboli

- Nel caso di codifica booleana dei simboli dell'alfabeto, due simboli differiscono per almeno uno dei valori con cui essi sono codificati. Supponiamo che una rete sequenziale abbia assegnato il suo alfabeto di ingresso codificato con n variabili di ingresso. Si possono, in generale, presentare i seguenti casi:

Tipologia delle variabili

- 1. tutte le n variabili di ingresso sono fra loro non correlate e indipendenti l'una dall'altra per cui il cambiamento di ciascuna di esse può far cambiare lo stato della rete. Esse sono dette a livello;
- 2. delle n variabili una di tipo impulsivo (clock), determina il momento della lettura delle n-1 variabili che con il loro valore definiscono con il loro valore la transizione da attivare. La variabile clock rappresenta un impulso che è generato sul fronte di salita o di discesa di un segnale (logica edge triggered). Si dice che l'attivazione avviene sul fronte di salita o di discesa della variabile di clock;

Tipologia delle variabili

- 3. tutte le variabili sono di tipo clock; ciascuna di esse genera con la propria commutazione di valore una transizione di stato. La rete è sincronizzata e ha tutte le variabili di sincronismo di tipo edge triggered;
- 4. delle n variabili una parte p è di tipo impulsivo (clock), la rimanente parte n-p a livello determina con il loro valore il tipo di transizione da attivare

Macchina impulsiva sincrona e asincrona

Metodologia di progetto di una macchina sequenziale

Dalla descrizione verbale al modello

Macchine completamente specificate

Equivalenza fra stati e fra macchine

Algoritmo di Paull-Ungar

Macchine parzialmente specificate

Inclusione di stati e macchine

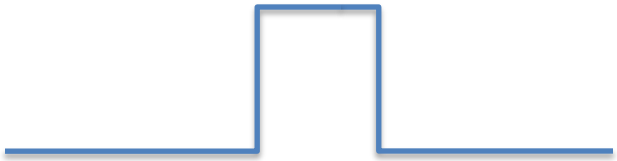
Stati compatibili

Copertura degli stati

Procedura di minimizzazione degli stati

Reti impulsive

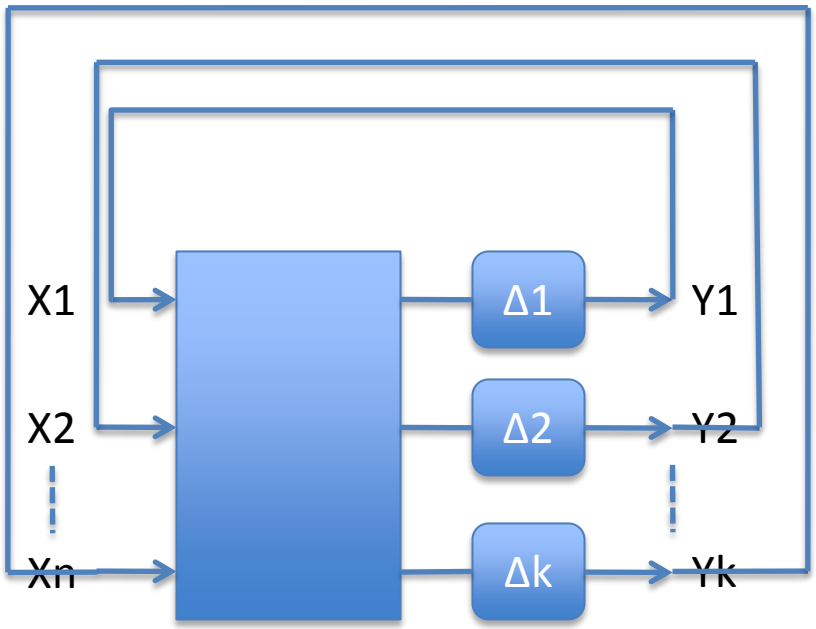
- Nel funzionamento delle reti di tipo impulsivo, occorre disporre di un simbolo i_0 (valore neutro) da utilizzare come sostegno dell'impulso tra ogni coppia di simboli significativi. Le reti impulsive necessitano, quindi, di almeno una variabile di ingresso di tipo clock

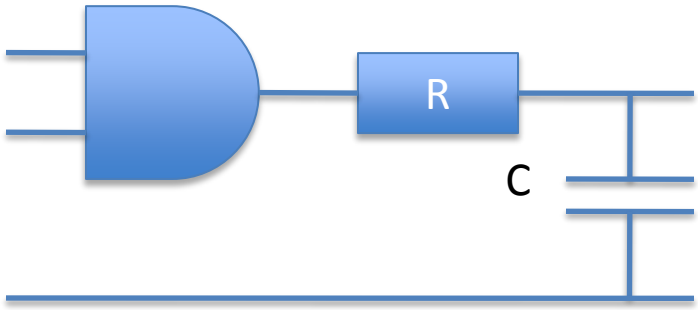


Sulle variabili di ingresso

- La variazione dello stato degli ingressi è determinato dall'epoca in cui accade l'evento e dalla tipologia dell'evento stesso. Un ingresso può codificare entrambe le cose con ciascuna delle n variabili x_i oppure associare alla variazione di una delle n variabili (clock) l'epoca di accadimento dell'evento e alle rimanenti n-1 variabili la codifica del tipo di evento
- Le $x_1, x_2, \ldots, x_n$ variabili di ingresso sono tra loro indipendenti e generano configurazioni tali da far cambiare stato alla rete. Se le variabili sono booleane, esse possono codificare uno spazio di 2^n configurazioni dello stato degli ingressi;
- $x_1, x_2, \ldots, x_{n-1}$ variabili sono dedicate alla codifica del tipo di evento, una variabile x_n, detta clock, determina con la sua variazione di fronte ($1 \rightarrow 0$ oppure $0 \rightarrow 1$) l'epoca di accadimento dell'evento codificato;
- Tutte le variabili sono di tipo clock e ciascuna di esse definisce con la propria commutazione una transizione di stato. In tale caso la rete è detta sincronizzata con variabili di sincronismo di tipo edge-triggered
- P variabili definiscono con la loro commutazione 0-1 il consenso a cambiare stato, mentre le rimanenti n-p variabili definiscono con il loro valore la transizione da attivare. La rete è, in tal caso, detta sincronizzata e ha n-p variabili a livello e p variabili di sincronismo di tipo edge triggered

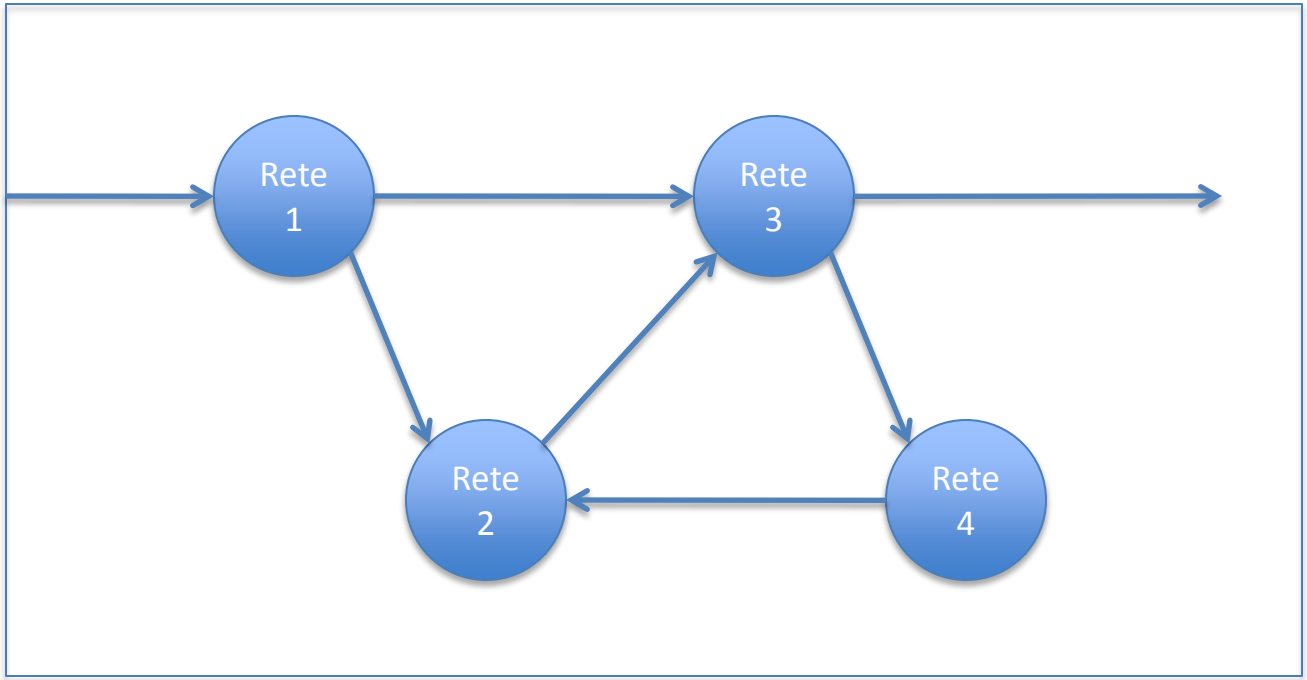
C




R
C

Sistemi di reti sequenziali

Esempio di interconnessione fra reti sequenziali



Ciascuna rete sequenziale può essere

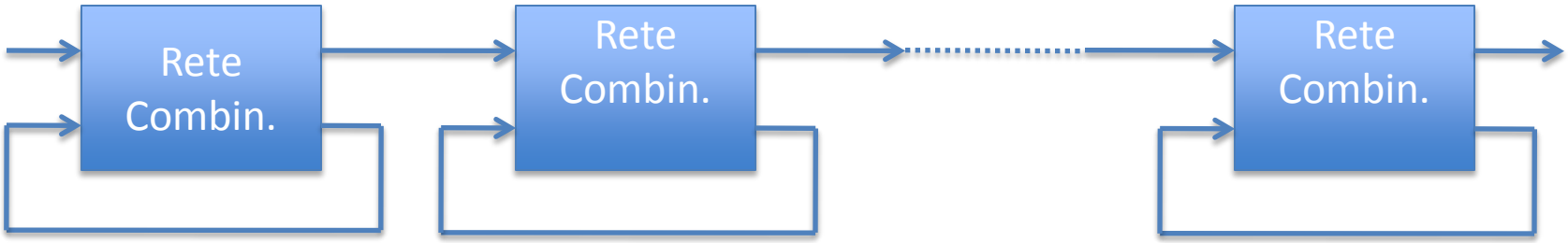
- Una rete di Mealy
- Una rete di Moore
- Ciascuna rete può essere di tipo impulsivo o asincrona
- Se le reti sono di tipo differente la descrizione del funzionamento del sistema dipende dal tipo di ogni singola rete e dalla loro interconnessione
- Se le reti sono omogenee e di tipo LLC è possibile descrivere il loro comportamento in modo sistematico con metodi di specifica e verifica ben consolidati nel mondo industriale (i sistemi digitali complessi sono di questo tipo).

Determinazione del tempo di ciclo

- In una rete level input, level output, clocked (LLC) occorre determinare il periodo del clock, ovvero il tempo di ciclo (intervallo minimo tra due impulsi di abilitazione della rete)
- Il tempo di ciclo dipende dalla tipologia delle macchine adottate (Mealy, Moore) e dalla topologia della loro interconnessione

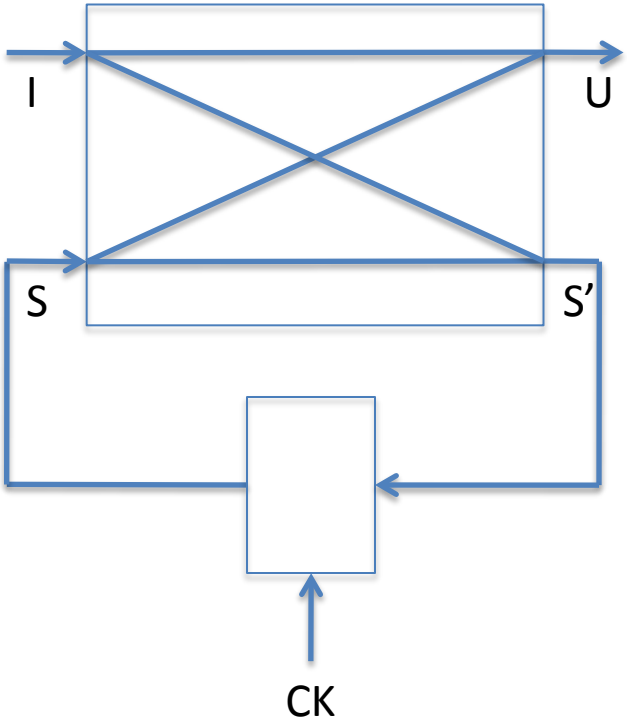
Catene aperte e catene chiuse di reti sequenziali

- Una catena aperta è costituita da una cascata di reti sequenziali in cui l'uscita dell'una è applicata all'ingresso della successiva, fatta eccezione della prima e dell'ultima

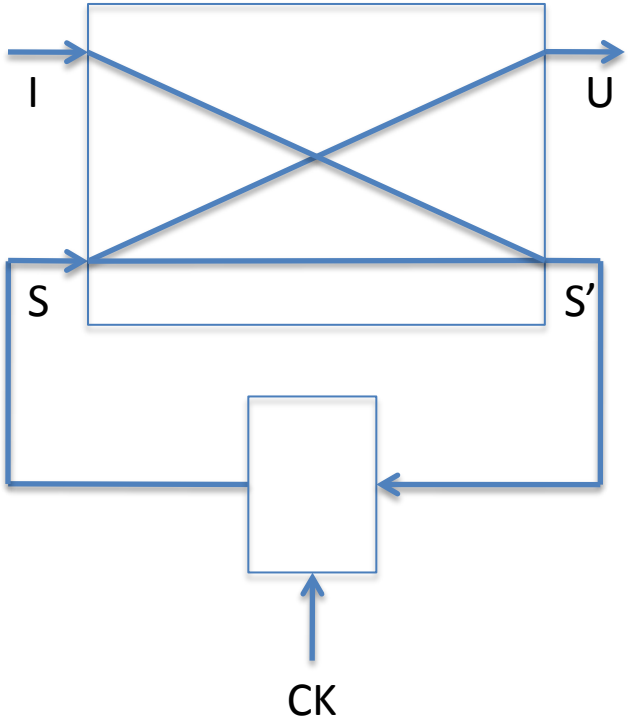


Reti di Mealy e di Moore

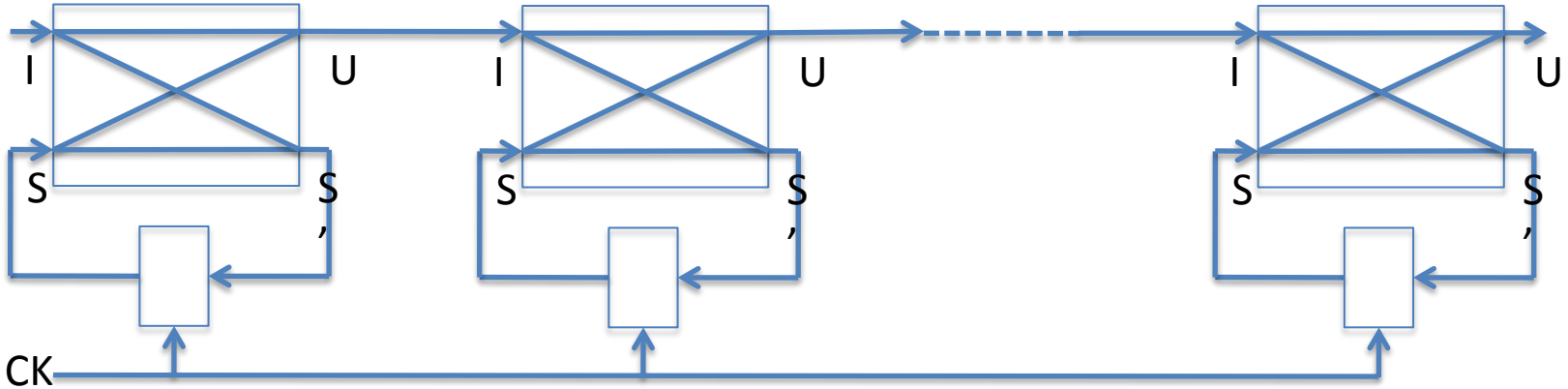
Rete di Mealy



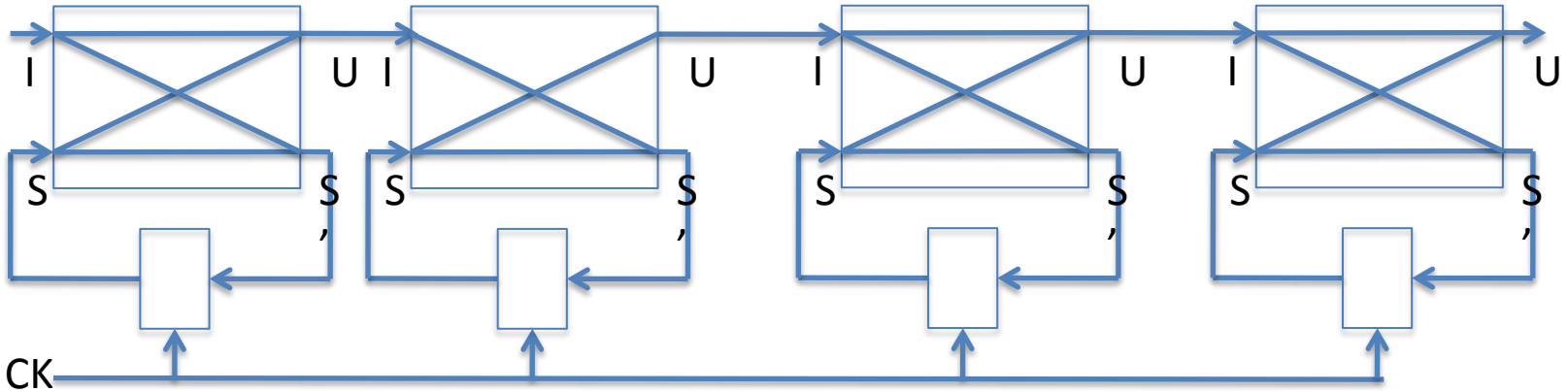
Rete di Moore



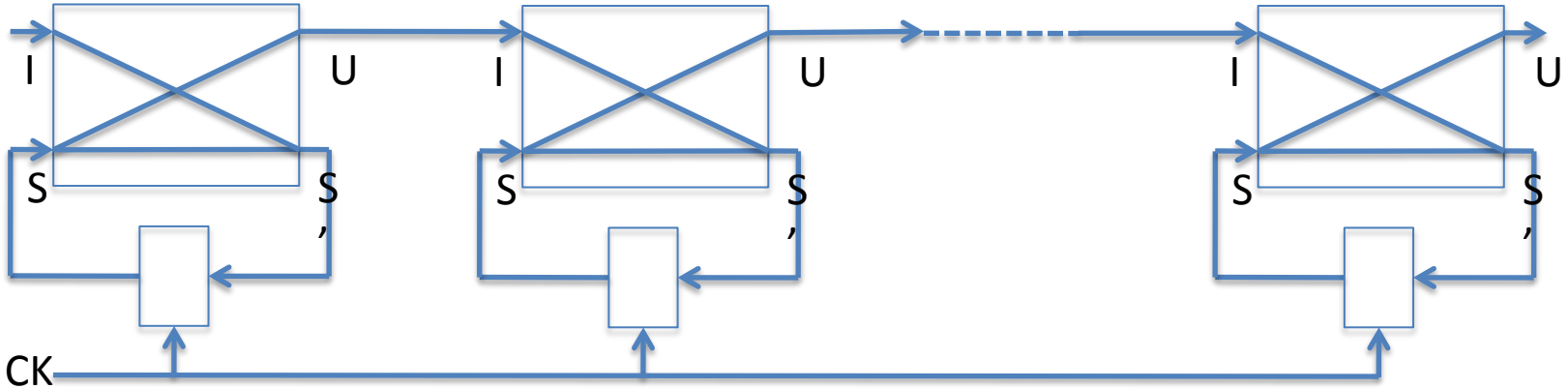
Catena aperta di macchine di Mealy



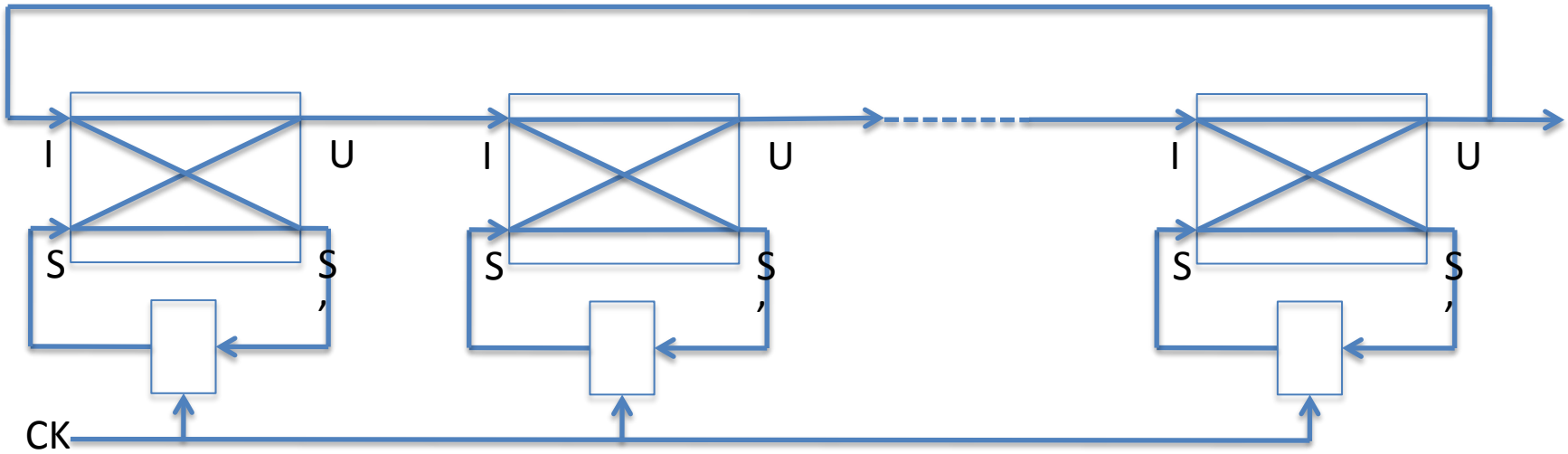
Catena aperta di macchine di Mealy-Moore



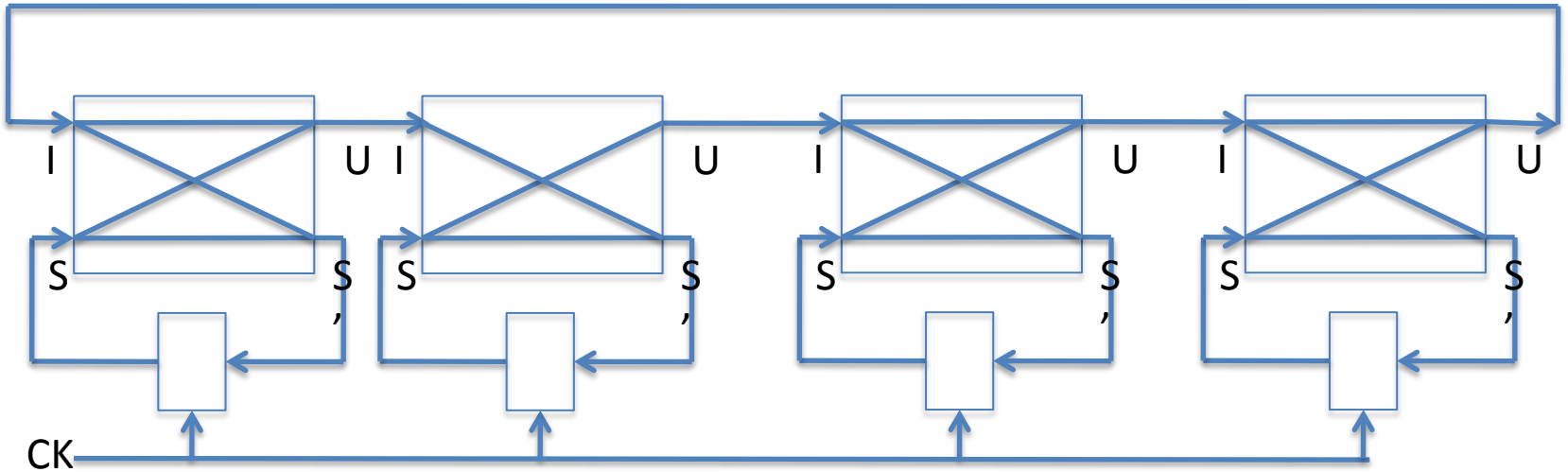
Catena aperta di macchine di Moore



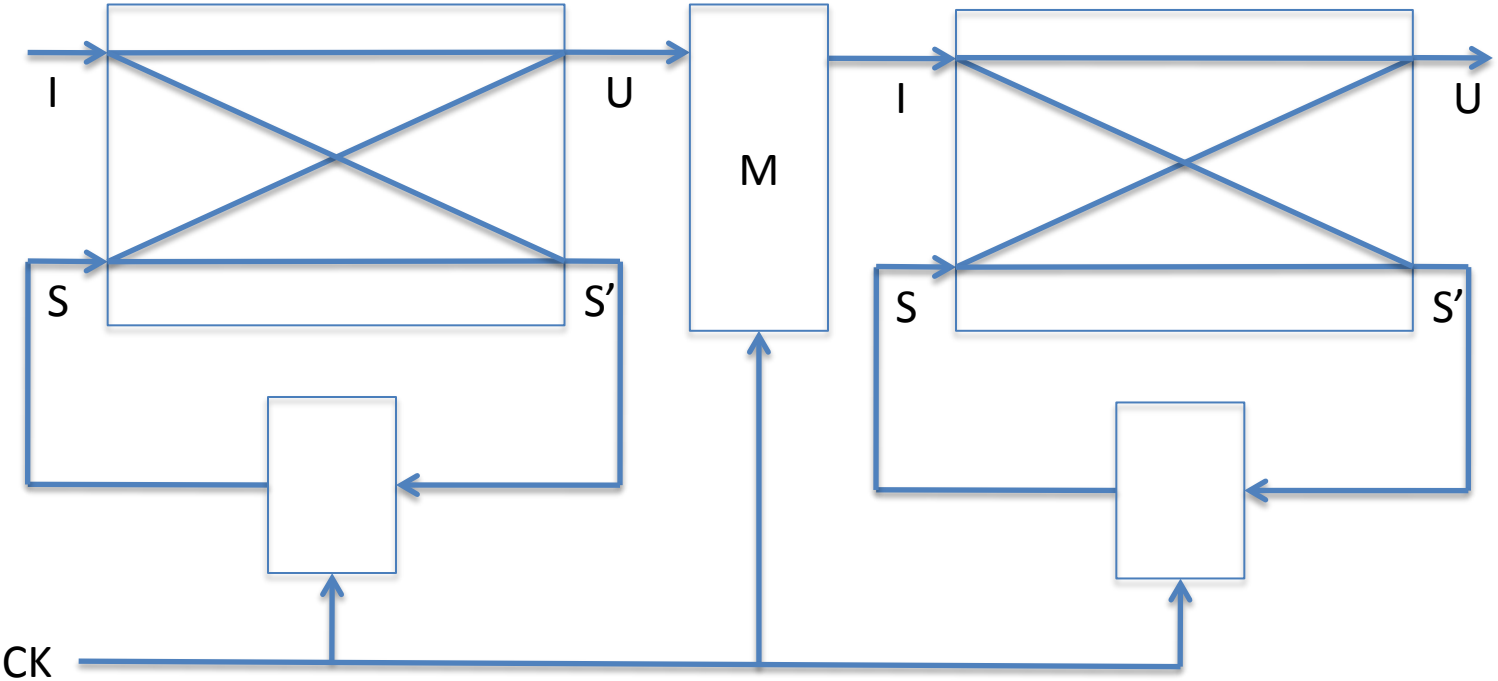
Catena chiusa di macchine di Mealy



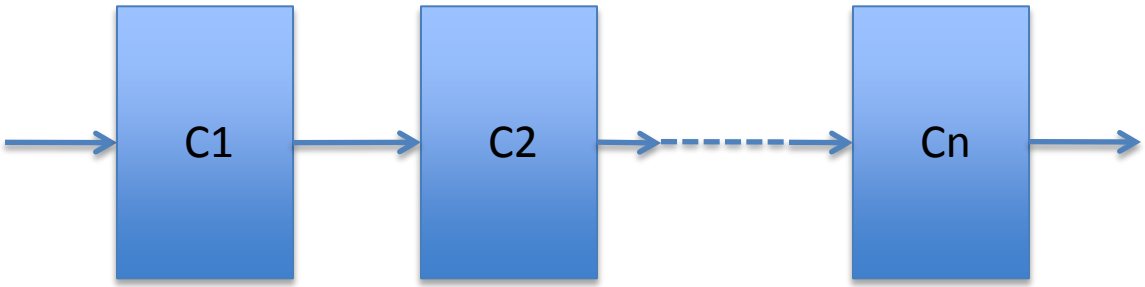
Catena chiusa Mealy-Moore



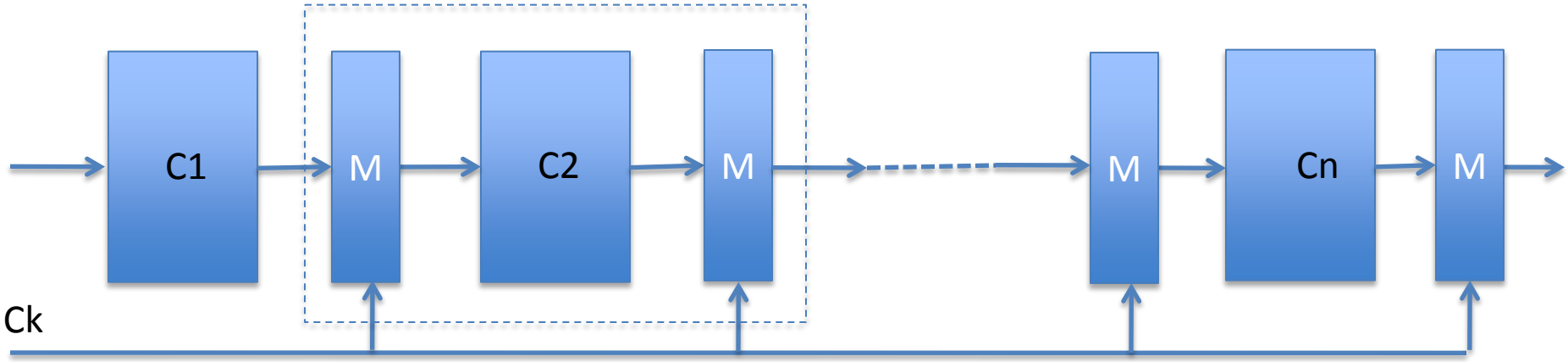
Catena chiusa di macchine di Mealy con registro



Catena aperta di reti combinatorie



Pipeline di reti combinatorie



- Tempo di ciclo T deve soddisfare la condizione:

$$T \geq t_f + \tau_c + t_{setup}$$

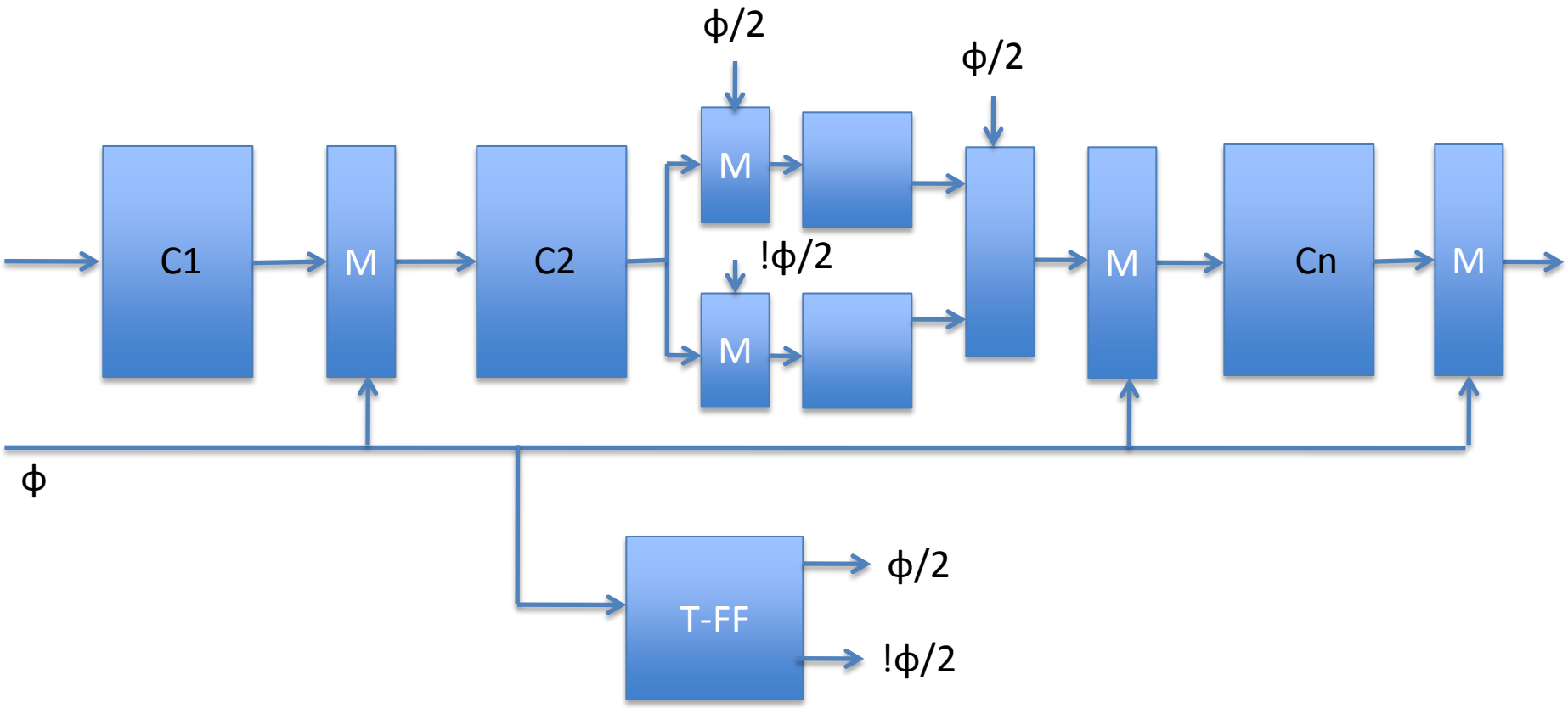
In cui

t_f è il tempo di commutazione del registro

τ_c è il tempo di calcolo della rete

t_{setup} è il tempo di set-up del registro

Architettura pipeline con reti parallelo



Architetture pipeline parallele

- 3 reti con tempi di calcolo di 50 ns e una rete con tempo di calcolo di 100 ns
  - Frequenza di pipe non può essere migliore di 100 MHz (T=1/f=100 ns) e le tre reti più veloci sono sfruttate al 50%
- Soluzione parallela che duplica rete a 100 ns, con reti funzionanti in controfase, multiplate nel tempo

Calcolo del tempo di ciclo

- T per la rete lenta è vincolato da

$$2T \geq t_f + 2\tau_{cmax} + \tau_{mux} + t_{setup}$$

da cui $$T \geq \tau_{cmax} + \tfrac{1}{2}(t_f + 2\tau_{mux} + t_{setup})$$

- T per la rete veloce è vincolato da

$$T \geq t_f + \tau_{cmax} + t_{setup}$$

- Va scelto il massimo fra i due:

$$T \geq \tau_{cmax} + \max(\tfrac{1}{2}(t_f + \tau_{mux} + t_{setup}), (t_f + t_{setup}))$$

Microlinguaggi M, PS, TS

- $\mu_h | O_v, \mu_{h+1}$ (11)
- $\mu_h | O_v, \mu_k$ (12)
- $\mu_h | O_0(Cr)\mu_{h+1}; (Cr)\ \mu_k$ (13)
- $\mu h | (C_1)O^1_j,\mu^1_k \quad ; (C_2)O^2_j,\mu^2_k;\ldots; (C_q)O^q_j,\mu^q_k;$ (14)
- $\mu_h | O_j(C_1)\ \mu^1_k\ ; (C_2)\ \mu^2_k;\ldots; (C_q)\ \mu^q_k$ (17)

Procedure di microprogrammazione

- Obiettivo delle procedure
  - Ridurre al minimo i tempi elementari in cui PO rimane inattiva durante l'esecuzione delle operazioni (il minimo è dipendente anche dal linguaggio usato)

Microlinguaggio M

- Si considera un blocco alla volta del diagramma di flusso
- If (blocco=blocco di decisione) then
  - $\mu_h | O_0(Cr)\mu_{h+1}$; (Cr) μ
- If (

Microprogrammazione MIC1

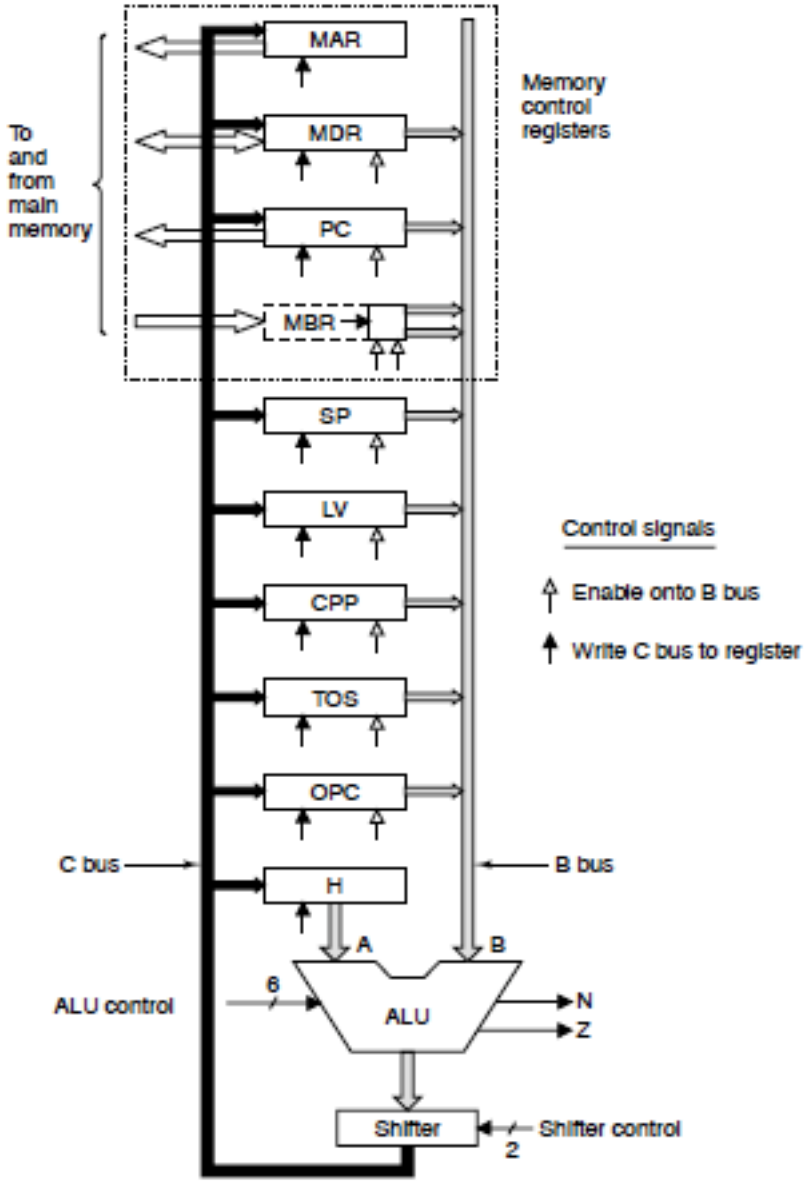
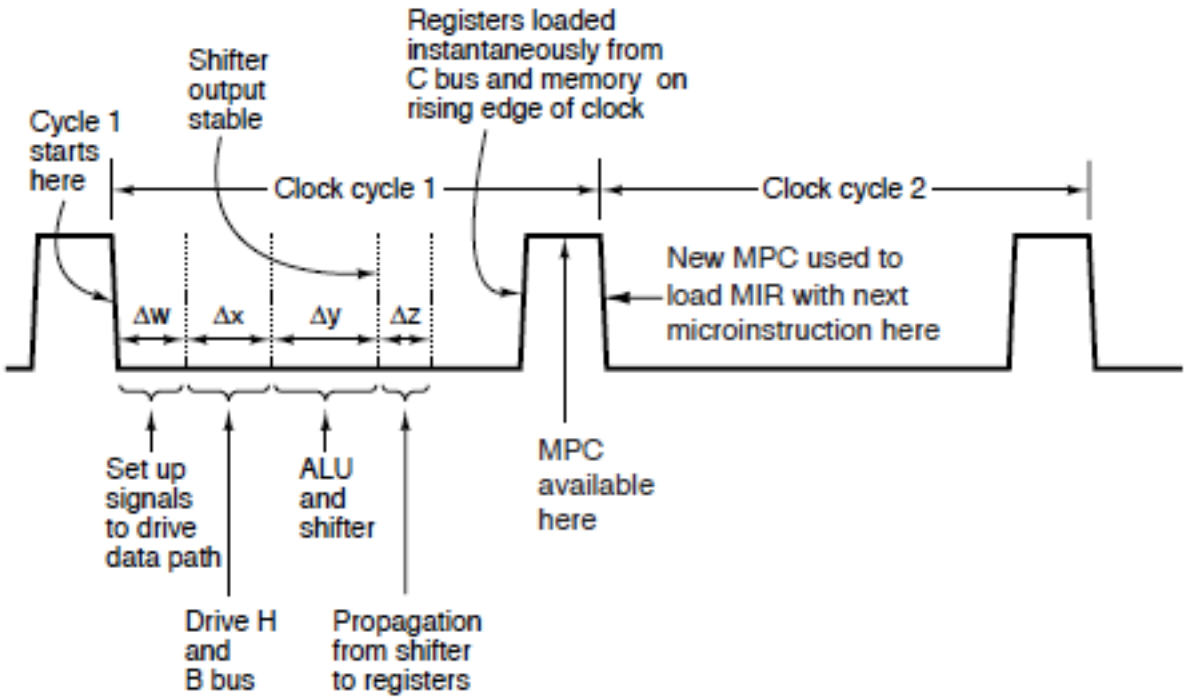


Figure 4-1. The data path of the example microarchitecture used in this chapter.

F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	1	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	–1

Figure 4-2. Useful combinations of ALU signals and the function performed.


Registers loaded instantaneously from C bus and memory on rising edge of clock
Shifter output stable
Cycle 1 starts here
Clock cycle 1
Clock cycle 2
New MPC used to load MIR with next microinstruction here
Δw
Δx
Δy
Δz
Set up signals to drive data path
Drive H and B bus
ALU and shifter
Propagation from shifter to registers
MPC available here

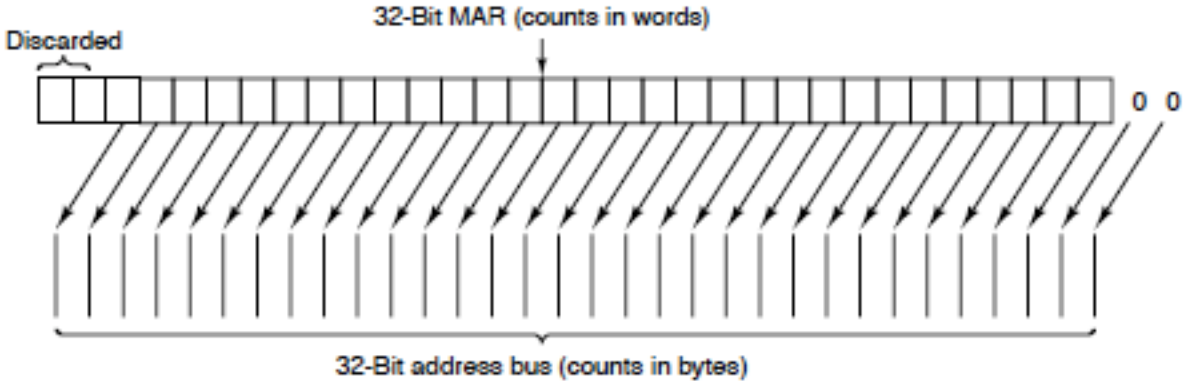


Figure 4-4. Mapping of the bits in MAR to the address bus.

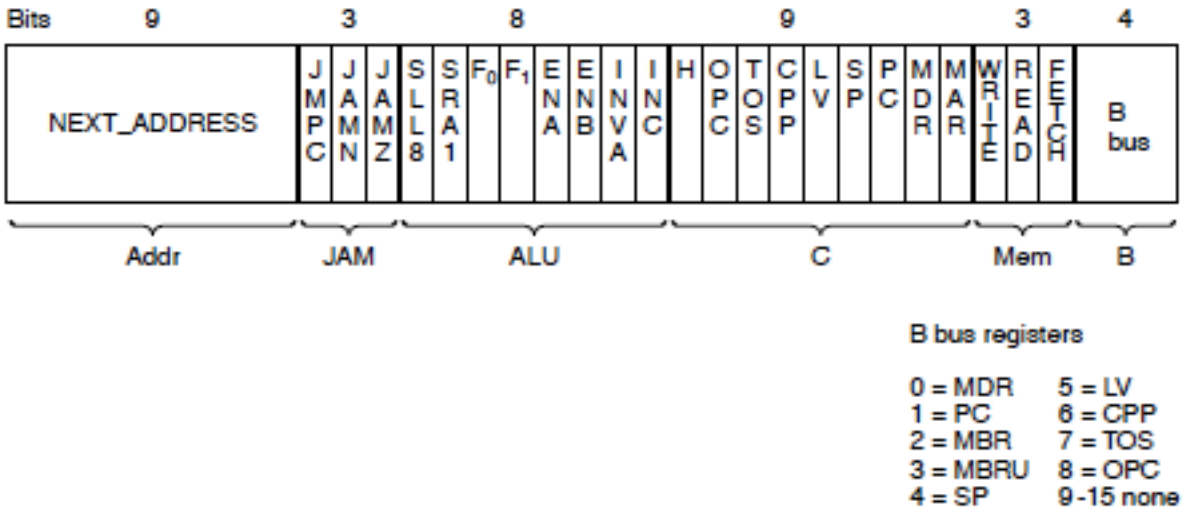


Figure 4-5. The microinstruction format for the Mic-1.

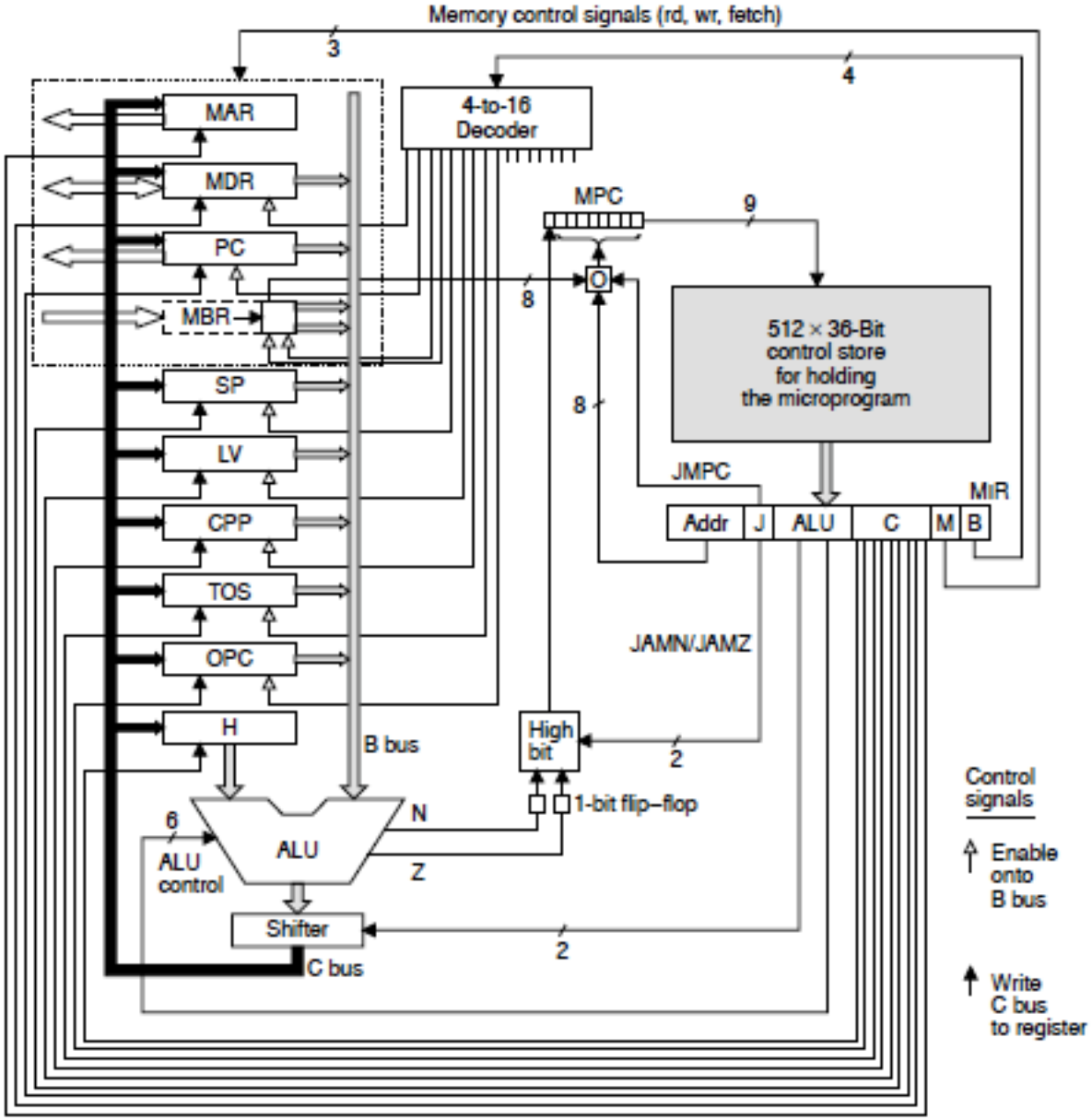


Figure 4-6. The complete block diagram of our example microarchitecture, the Mic-1.

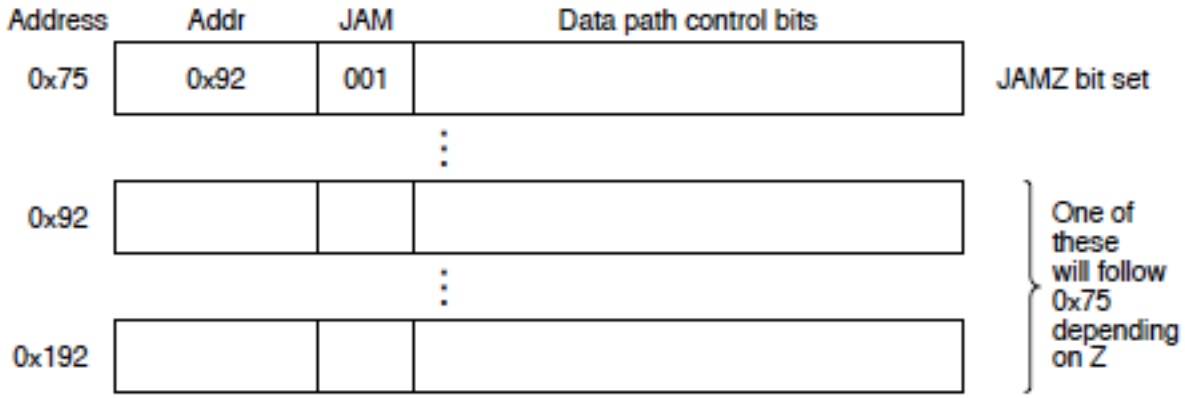


Figure 4-7. A microinstruction with JAMZ set to 1 has two potential successors.

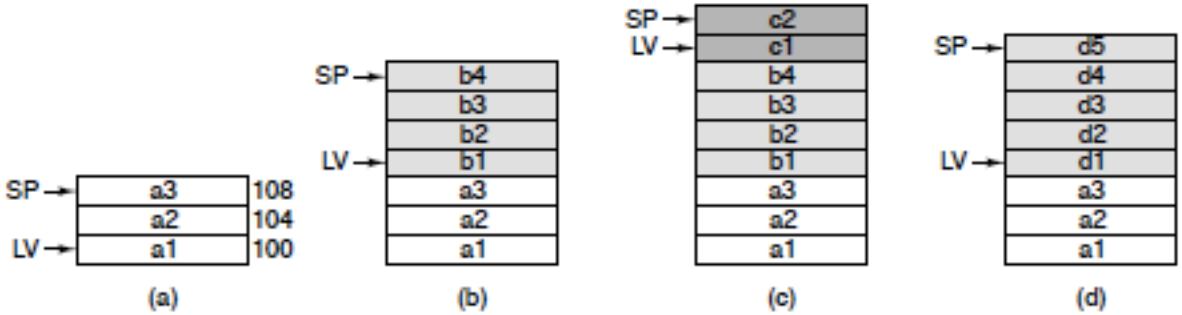


Figure 4-8. Use of a stack for storing local variables. (a) While *A* is active. (b) After *A* calls *B*. (c) After *B* calls *C*. (d) After *C* and *B* return and *A* calls *D*.

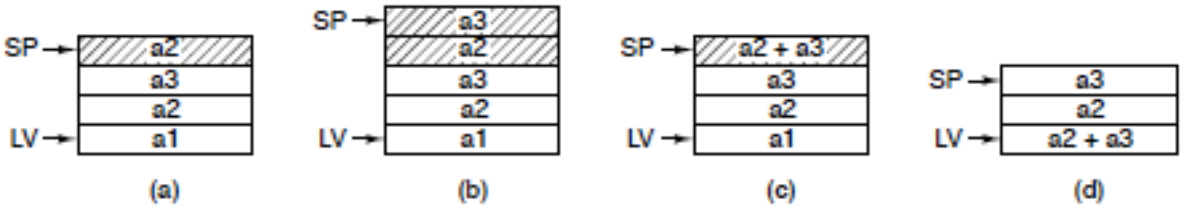


Figure 4-9. Use of an operand stack for doing an arithmetic computation.

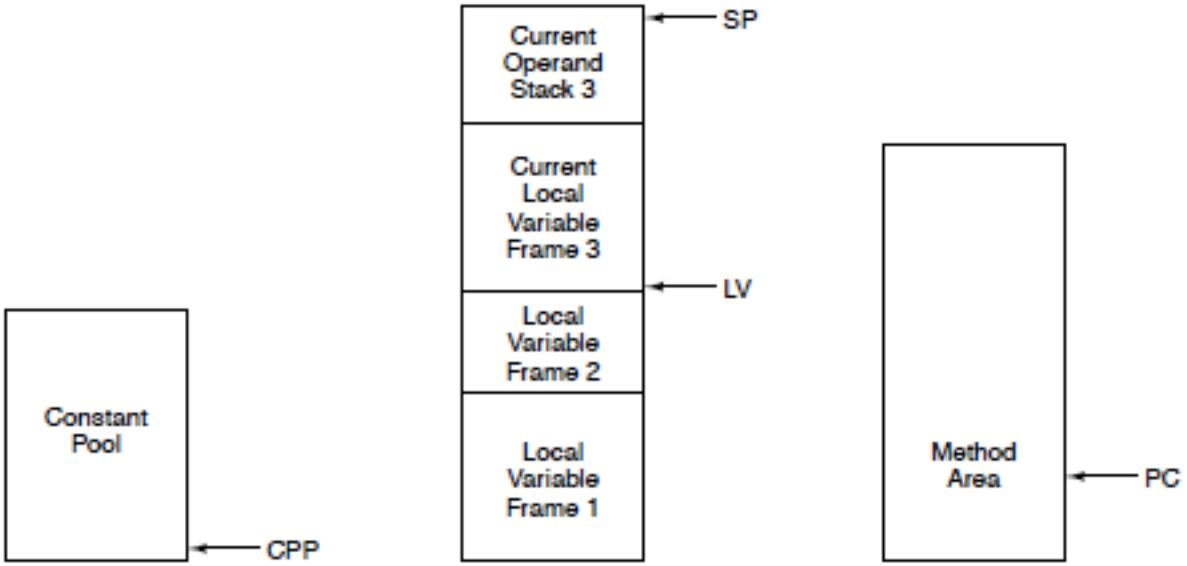


Figure 4-10. The various parts of the IJVM memory.

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

Figure 4-11. The IJVM instruction set. The operands *byte*, *const*, and *varnum* are 1 byte. The operands *disp*, *index*, and *offset* are 2 bytes.

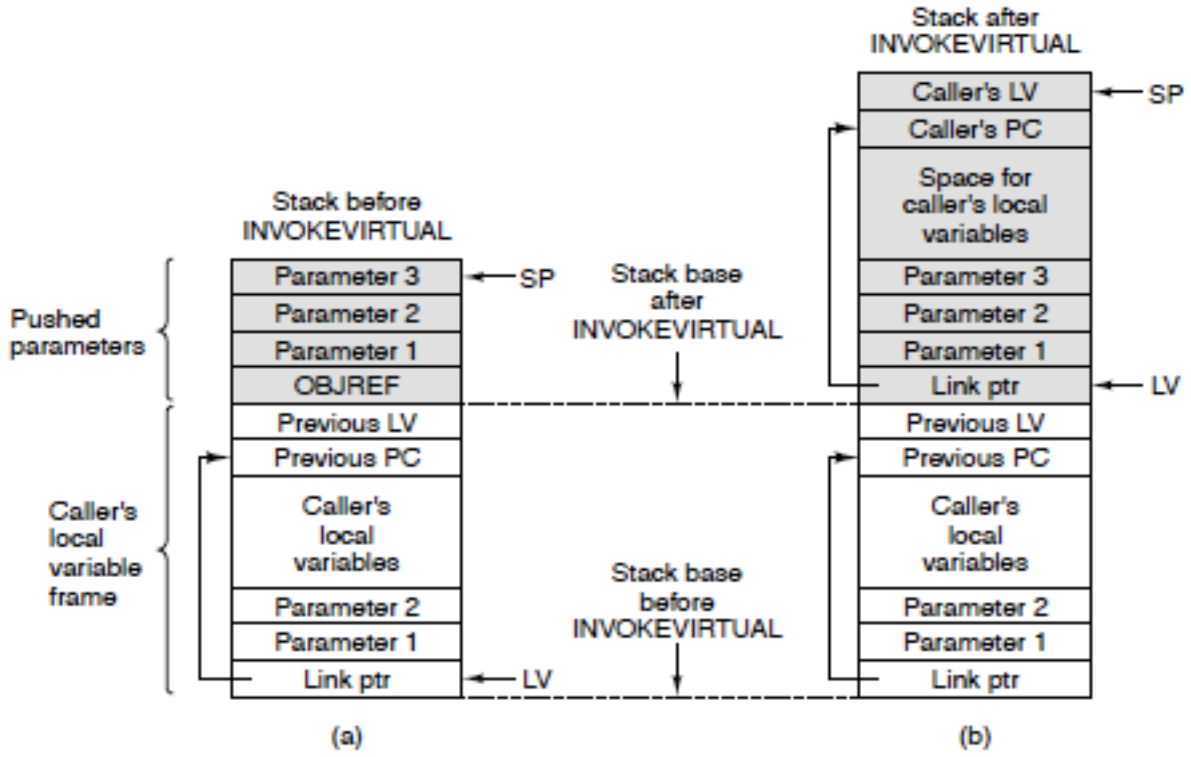


Figure 4-12. (a) Memory before executing INVOKEVIRTUAL. (b) After executing it.

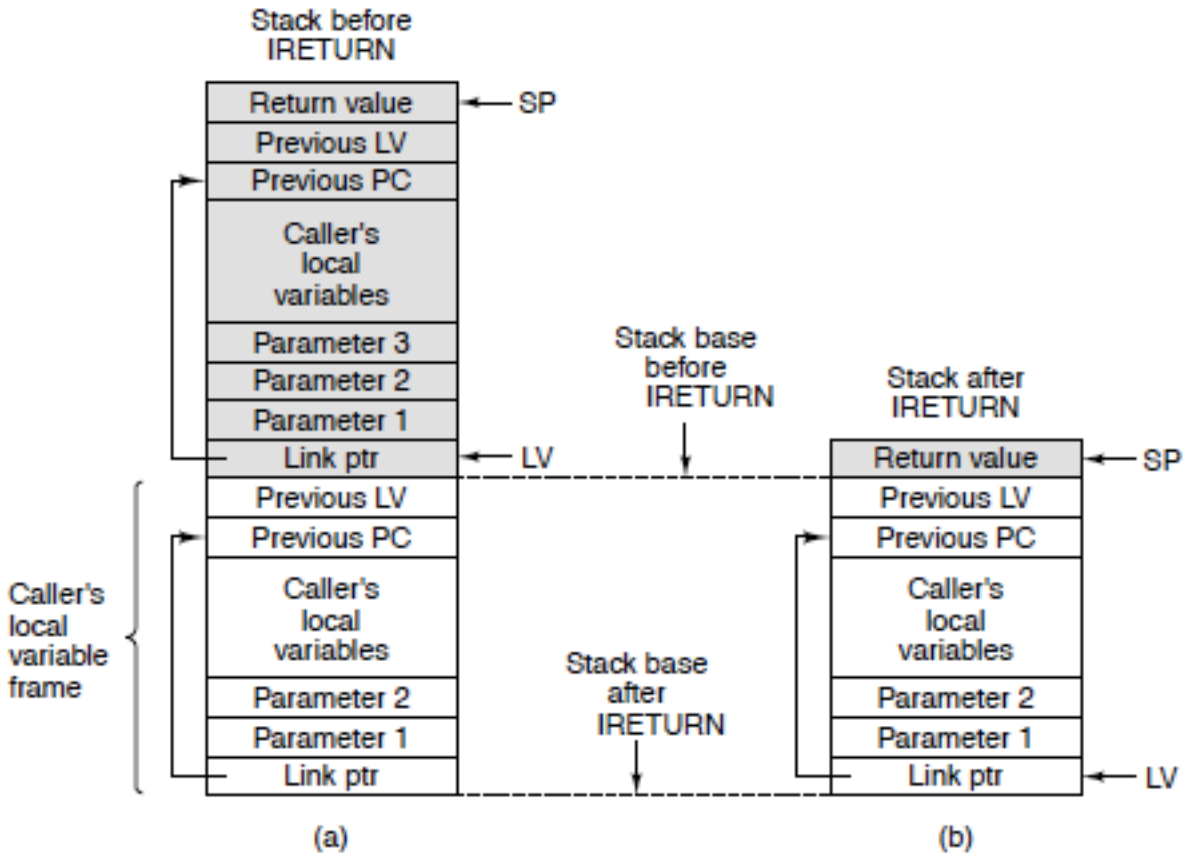


Figure 4-13. (a) Memory before executing IRETURN. (b) After executing it.

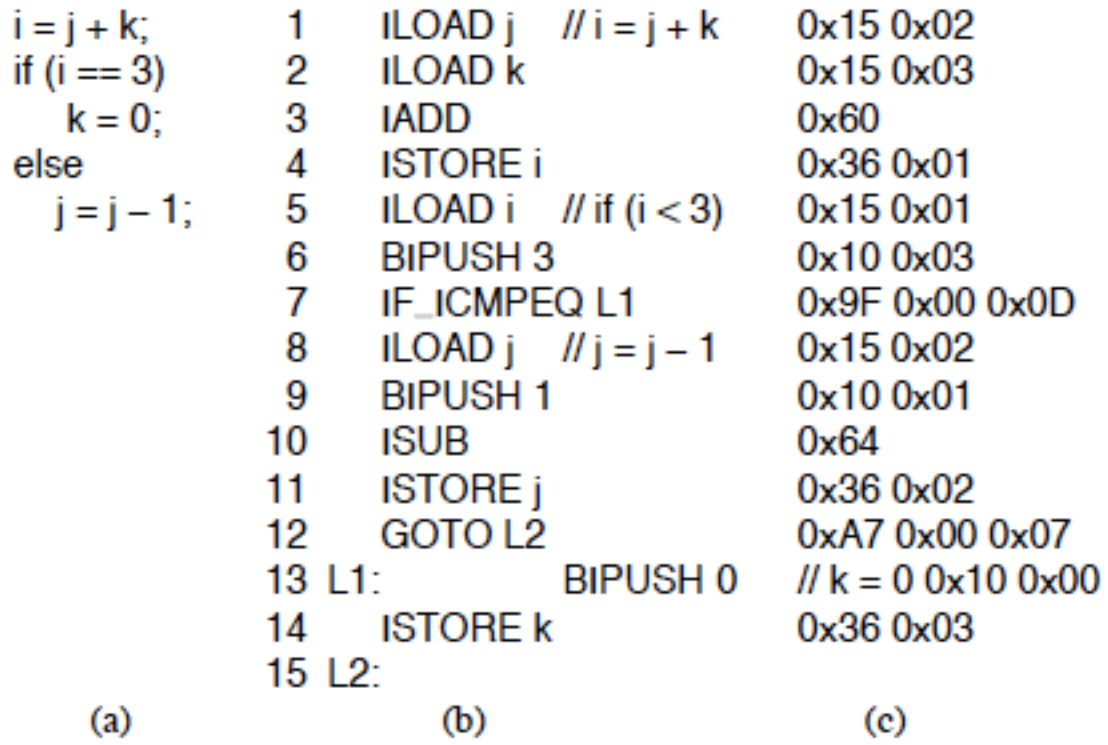


Figure 4-14. (a) A Java fragment. (b) The corresponding Java assembly language. (c) The IJVM program in hexadecimal.

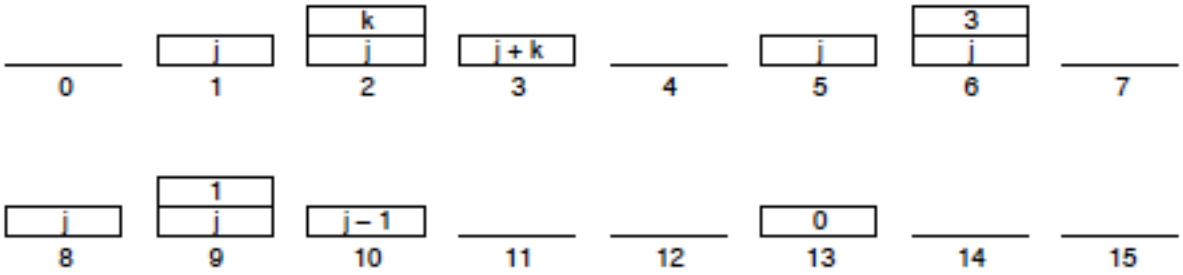


Figure 4-15. The stack after each instruction of Fig. 4-14(b).

DEST = H
DEST = SOURCE
DEST = $\overline{H}$
DEST = $\overline{SOURCE}$
DEST = H + SOURCE
DEST = H + SOURCE + 1
DEST = H + 1
DEST = SOURCE + 1
DEST = SOURCE – H
DEST = SOURCE – 1
DEST = –H
DEST = H AND SOURCE
DEST = H OR SOURCE
DEST = 0
DEST = 1
DEST = –1

Figure 4-16. All permitted operations. Any of the above operations may be extended by adding "<< 8" to them to shift the result left by 1 byte. For example, a common operation is H = MBR << 8

Label	Operations	Comments
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch
nop1	goto Main1	Do nothing
iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack
isub1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
isub2	H = TOS	H = top of stack
isub3	MDR = TOS = MDR – H; wr; goto Main1	Do subtraction; write to top of stack
iand1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iand2	H = TOS	H = top of stack
iand3	MDR = TOS = MDR AND H; wr; goto Main1	Do AND; write to new top of stack
ior1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
ior2	H = TOS	H = top of stack
ior3	MDR = TOS = MDR OR H; wr; goto Main1	Do OR; write to new top of stack
dup1	MAR = SP = SP + 1	Increment SP and copy to MAR
dup2	MDR = TOS; wr; goto Main1	Write new stack word
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
pop2		Wait for new TOS to be read from memory
pop3	TOS = MDR; goto Main1	Copy new word to TOS
swap1	MAR = SP – 1; rd	Set MAR to SP – 1; read 2nd word from stack
swap2	MAR = SP	Set MAR to top word
swap3	H = MDR; wr	Save TOS in H; write 2nd word to top of stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP – 1; wr	Set MAR to SP – 1; write as 2nd word on stack
swap6	TOS = H; goto Main1	Update TOS
bipush1	SP = MAR = SP + 1	MBR = the byte to push onto stack
bipush2	PC = PC + 1; fetch	Increment PC, fetch next opcode
bipush3	MDR = TOS = MBR; wr; goto Main1	Sign-extend constant and push on stack
iload1	H = LV	MBR contains index; copy LV to H
iload2	MAR = MBRU + H; rd	MAR = address of local variable to push
iload3	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload4	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload5	TOS = MDR; goto Main1	Update TOS
istore1	H = LV	MBR contains index; Copy LV to H
istore2	MAR = MBRU + H	MAR = address of local variable to store into
istore3	MDR = TOS; wr	Copy TOS to MDR; write word
istore4	SP = MAR = SP – 1; rd	Read in next-to-top word on stack
istore5	PC = PC + 1; fetch	Increment PC; fetch next opcode
istore6	TOS = MDR; goto Main1	Update TOS

wide1	PC = PC + 1; fetch; goto (MBR OR 0x100)	Multiway branch with high bit set
wide_iload1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd
wide_iload2	H = MBRU << 8	H = 1st index byte shifted left 8 bits
wide_iload3	H = MBRU OR H	H = 16-bit index of local variable
wide_iload4	MAR = LV + H; rd; goto iload3	MAR = address of local variable to push
wide_istore1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd
wide_istore2	H = MBRU << 8	H = 1st index byte shifted left 8 bits
wide_istore3	H = MBRU OR H	H = 16-bit index of local variable
wide_istore4	MAR = LV + H; goto istore3	MAR = address of local variable to store into
ldc_w1	PC = PC + 1; fetch	MBR contains 1st index byte; fetch 2nd
ldc_w2	H = MBRU << 8	H = 1st index byte << 8
ldc_w3	H = MBRU OR H	H = 16-bit index into constant pool
ldc_w4	MAR = H + CPP; rd; goto iload3	MAR = address of constant in pool

Figure 4-17. The microprogram for the Mic-1 (part 1 of 3).

Label	Operations	Comments
iinc1	H = LV	MBR contains index; Copy LV to H
iinc2	MAR = MBRU + H; rd	Copy LV + index to MAR; Read variable
iinc3	PC = PC + 1; fetch	Fetch constant
iinc4	H = MDR	Copy variable to H
iinc5	PC = PC + 1; fetch	Fetch next opcode
iinc6	MDR = MBR + H; wr; goto Main1	Put sum in MDR; update variable
goto1	OPC = PC – 1	Save address of opcode.
goto2	PC = PC + 1; fetch	MBR = 1st byte of offset; fetch 2nd byte
goto3	H = MBR << 8	Shift and save signed first byte in H
goto4	H = MBRU OR H	H = 16-bit branch offset
goto5	PC = OPC + H; fetch	Add offset to OPC
goto6	goto Main1	Wait for fetch of next opcode
iflt1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iflt2	OPC = TOS	Save TOS in OPC temporarily
iflt3	TOS = MDR	Put new top of stack in TOS
iflt4	N = OPC; if (N) goto T; else goto F	Branch on N bit
ifeq1	MAR = SP = SP – 1; rd	Read in next-to-top word of stack
ifeq2	OPC = TOS	Save TOS in OPC temporarily
ifeq3	TOS = MDR	Put new top of stack in TOS
ifeq4	Z = OPC; if (Z) goto T; else goto F	Branch on Z bit
if_icmpeq1	MAR = SP = SP – 1; rd	Read in next-to-top word of stack
if_icmpeq2	MAR = SP = SP – 1	Set MAR to read in new top-of-stack
if_icmpeq3	H = MDR; rd	Copy second stack word to H
if_icmpeq4	OPC = TOS	Save TOS in OPC temporarily
if_icmpeq5	TOS = MDR	Put new top of stack in TOS
if_icmpeq6	Z = OPC – H; if (Z) goto T; else goto F	If top 2 words are equal, goto T, else goto F
T	OPC = PC – 1; fetch; goto goto2	Same as goto1; needed for target address
F	PC = PC + 1	Skip first offset byte
F2	PC = PC + 1; fetch	PC now points to next opcode
F3	goto Main1	Wait for fetch of opcode

invokevirtual1	PC = PC + 1; fetch	MBR = index byte 1; inc. PC, get 2nd byte
invokevirtual2	H = MBRU << 8	Shift and save first byte in H
invokevirtual3	H = MBRU OR H	H = offset of method pointer from CPP
invokevirtual4	MAR = CPP + H; rd	Get pointer to method from CPP area
invokevirtual5	OPC = PC + 1	Save Return PC in OPC temporarily
invokevirtual6	PC = MDR; fetch	PC points to new method; get param count
invokevirtual7	PC = PC + 1; fetch	Fetch 2nd byte of parameter count
invokevirtual8	H = MBRU << 8	Shift and save first byte in H
invokevirtual9	H = MBRU OR H	H = number of parameters
invokevirtual10	PC = PC + 1; fetch	Fetch first byte of # locals
invokevirtual11	TOS = SP – H	TOS = address of OBJREF – 1
invokevirtual12	TOS = MAR = TOS + 1	TOS = address of OBJREF (new LV)
invokevirtual13	PC = PC + 1; fetch	Fetch second byte of # locals
invokevirtual14	H = MBRU << 8	Shift and save first byte in H
invokevirtual15	H = MBRU OR H	H = # locals
invokevirtual16	MDR = SP + H + 1; wr	Overwrite OBJREF with link pointer
invokevirtual17	MAR = SP = MDR;	Set SP, MAR to location to hold old PC
invokevirtual18	MDR = OPC; wr	Save old PC above the local variables
invokevirtual19	MAR = SP = SP + 1	SP points to location to hold old LV
invokevirtual20	MDR = LV; wr	Save old LV above saved PC
invokevirtual21	PC = PC + 1; fetch	Fetch first opcode of new method.
invokevirtual22	LV = TOS; goto Main1	Set LV to point to LV Frame

Figure 4-17. The microprogram for the Mic-1 (part 2 of 3).

Label	Operations	Comments
ireturn1	MAR = SP = LV; rd	Reset SP, MAR to get link pointer
ireturn2		Wait for read
ireturn3	LV = MAR = MDR; rd	Set LV to link ptr; get old PC
ireturn4	MAR = LV + 1	Set MAR to read old LV
ireturn5	PC = MDR; rd; fetch	Restore PC; fetch next opcode
ireturn6	MAR = SP	Set MAR to write TOS
ireturn7	LV = MDR	Restore LV
ireturn8	MDR = TOS; wr; goto Main1	Save return value on original top of stack

Figure 4-17. The microprogram for the Mic-1 (part 3 of 3).

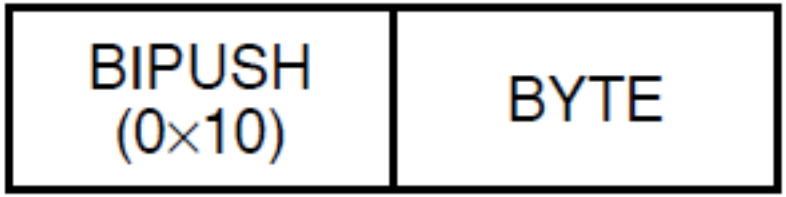


Figure 4-18. The BIPUSH instruction format.

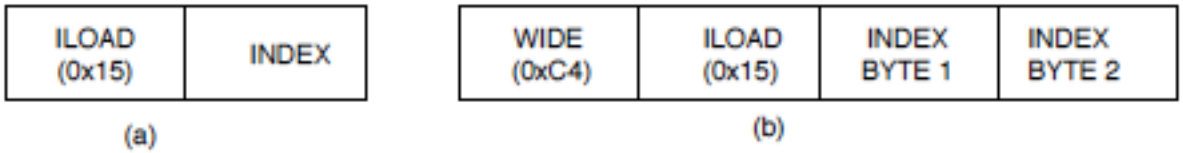


Figure 4-19. (a) ILOAD with a 1-byte index. (b) WIDE ILOAD with a 2-byte index.

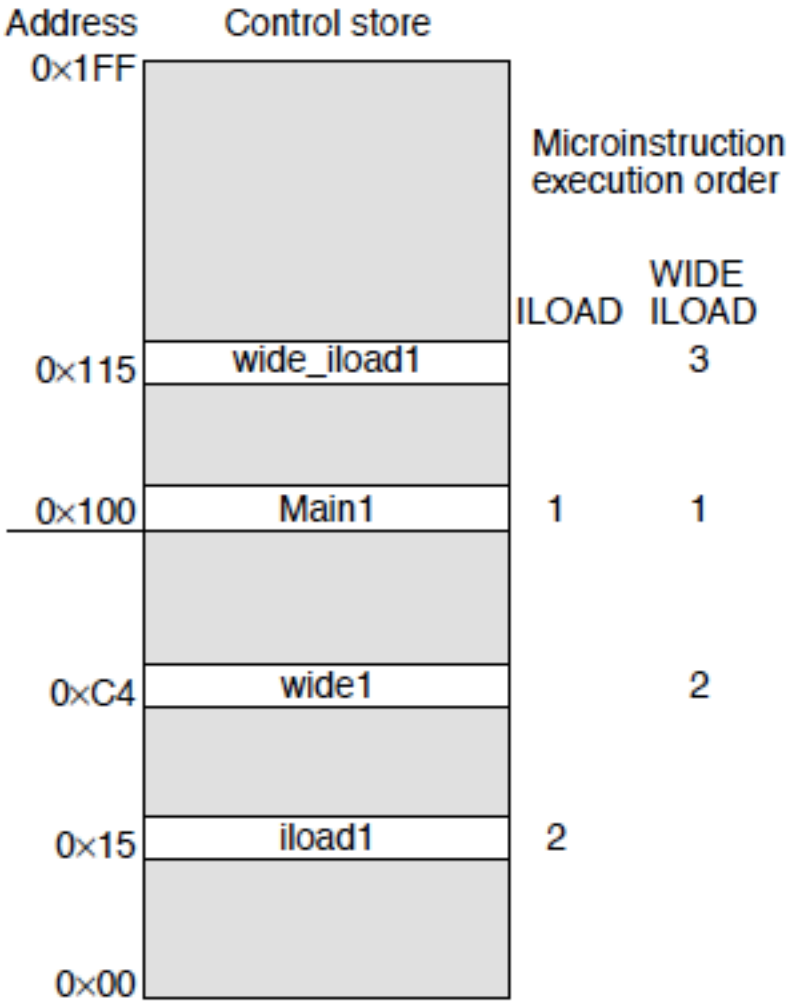


Figure 4-20. The initial microinstruction sequence for ILOAD and WIDE ILOAD. The addresses are examples.

Figure 4-21. The IINC instruction has two different operand fields.

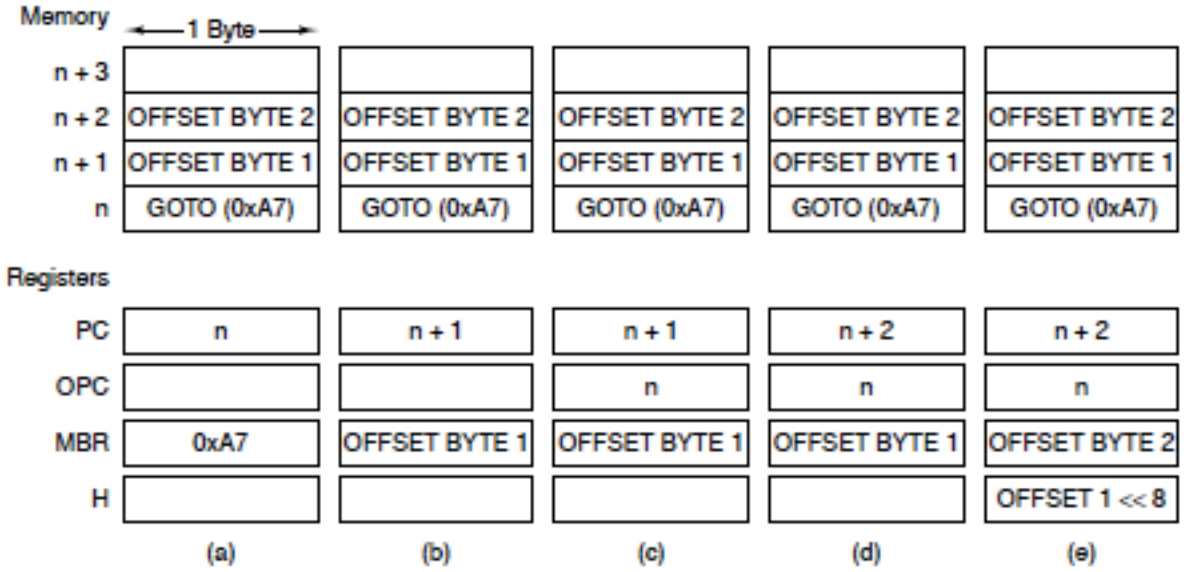


Figure 4-22. The situation at the start of various microinstructions. (a) Main1. (b) goto1. (c) goto2. (d) goto3. (e) goto4.

Label	Operations	Comments
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
pop2		Wait for new TOS to be read from memory
pop3	TOS = MDR; goto Main1	Copy new word to TOS
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch

Figure 4-23. New microprogram sequence for executing POP.

Label	Operations	Comments
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
Main1.pop	PC = PC + 1; fetch	MBR holds opcode; fetch next byte
pop3	TOS = MDR; goto (MBR)	Copy new word to TOS; dispatch on opcode

Figure 4-24. Enhanced microprogram sequence for executing POP.

Label	Operations	Comments
iload1	H = LV	MBR contains index; Copy LV to H
iload2	MAR = MBRU + H; rd	MAR = address of local variable to push
iload3	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload4	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload5	TOS = MDR; goto Main1	Update TOS
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch

Figure 4-25. Mic-1 code for executing ILOAD.

Label	Operations	Comments
iload1	MAR = MBRU + LV; rd	MAR = address of local variable to push
iload2	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload3	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload4	TOS = MDR	Update TOS
iload5	PC = PC + 1; fetch; goto (MBR)	MBR already holds opcode; fetch index byte

Figure 4-26. Three-bus code for executing ILOAD.

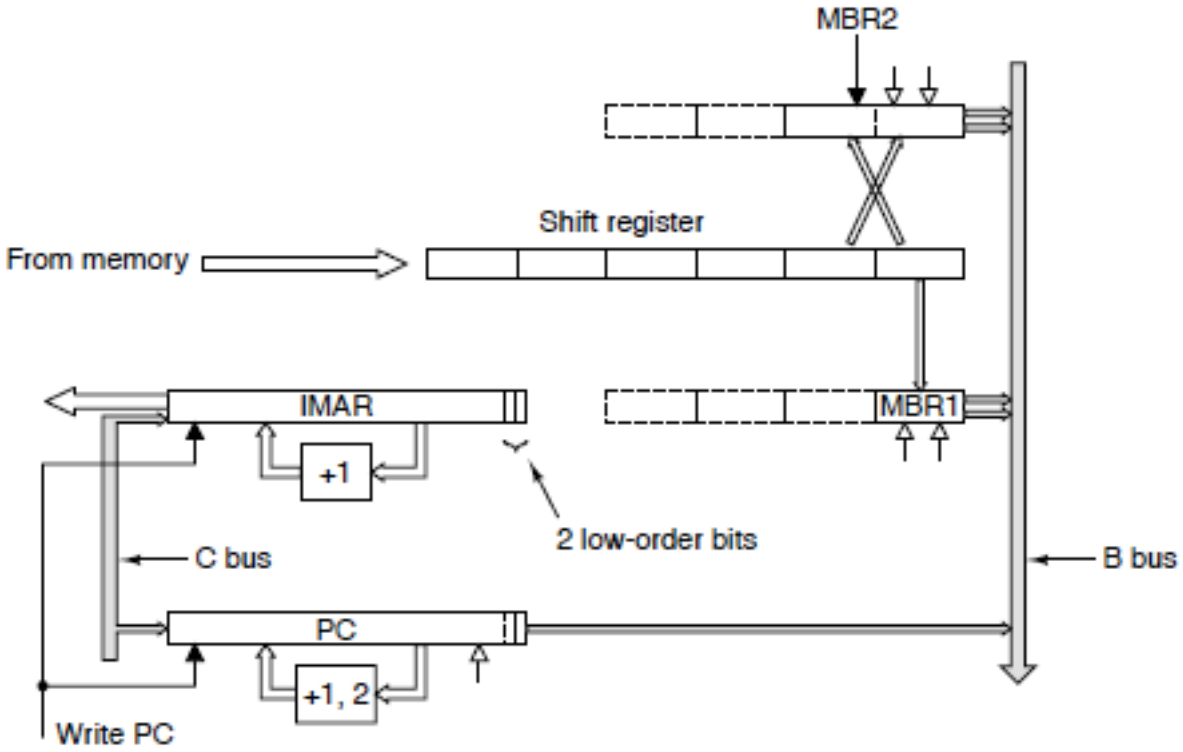
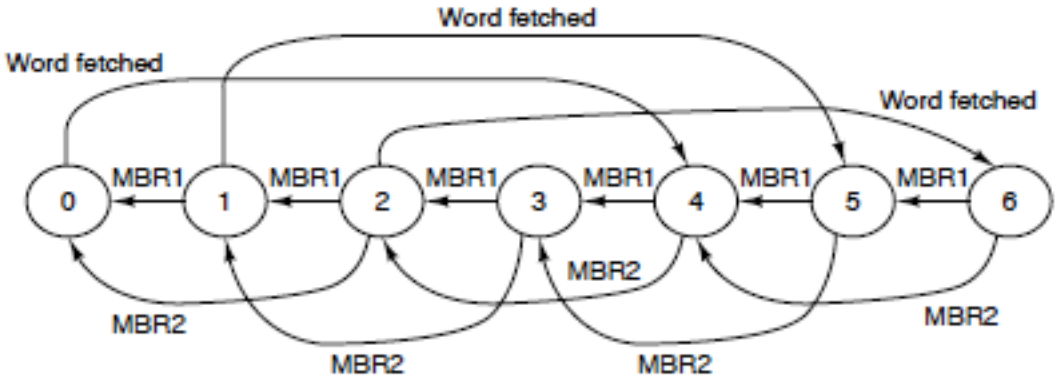


Figure 4-27. A fetch unit for the Mic-1.



Transitions

MBR1: Occurs when MBR1 is read
MBR2: Occurs when MBR2 is read
Word fetched: Occurs when a memory word is read and 4 bytes are put into the shift register

Figure 4-28. A finite state machine for implementing the IFU.

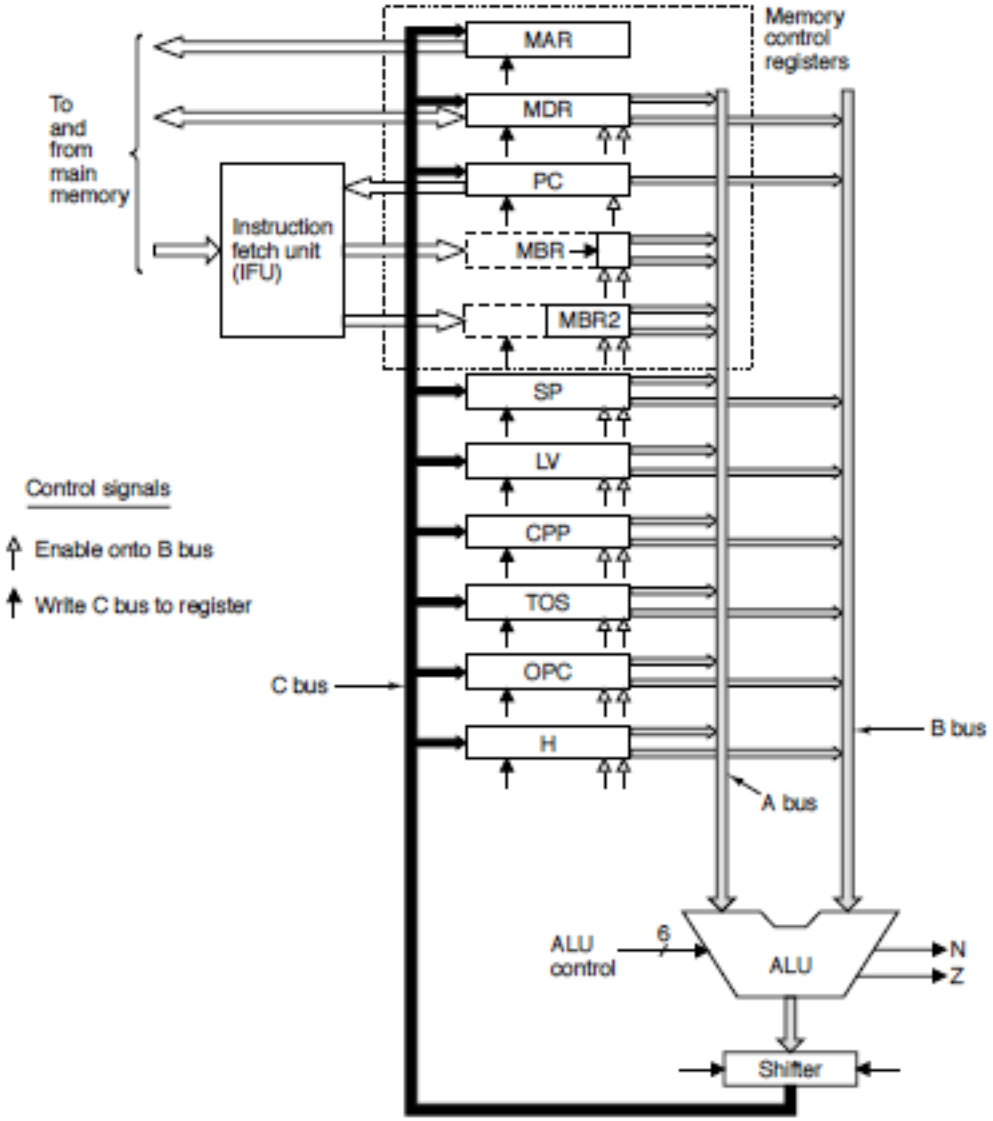


Figure 4-29. The datapath for Mic-2.

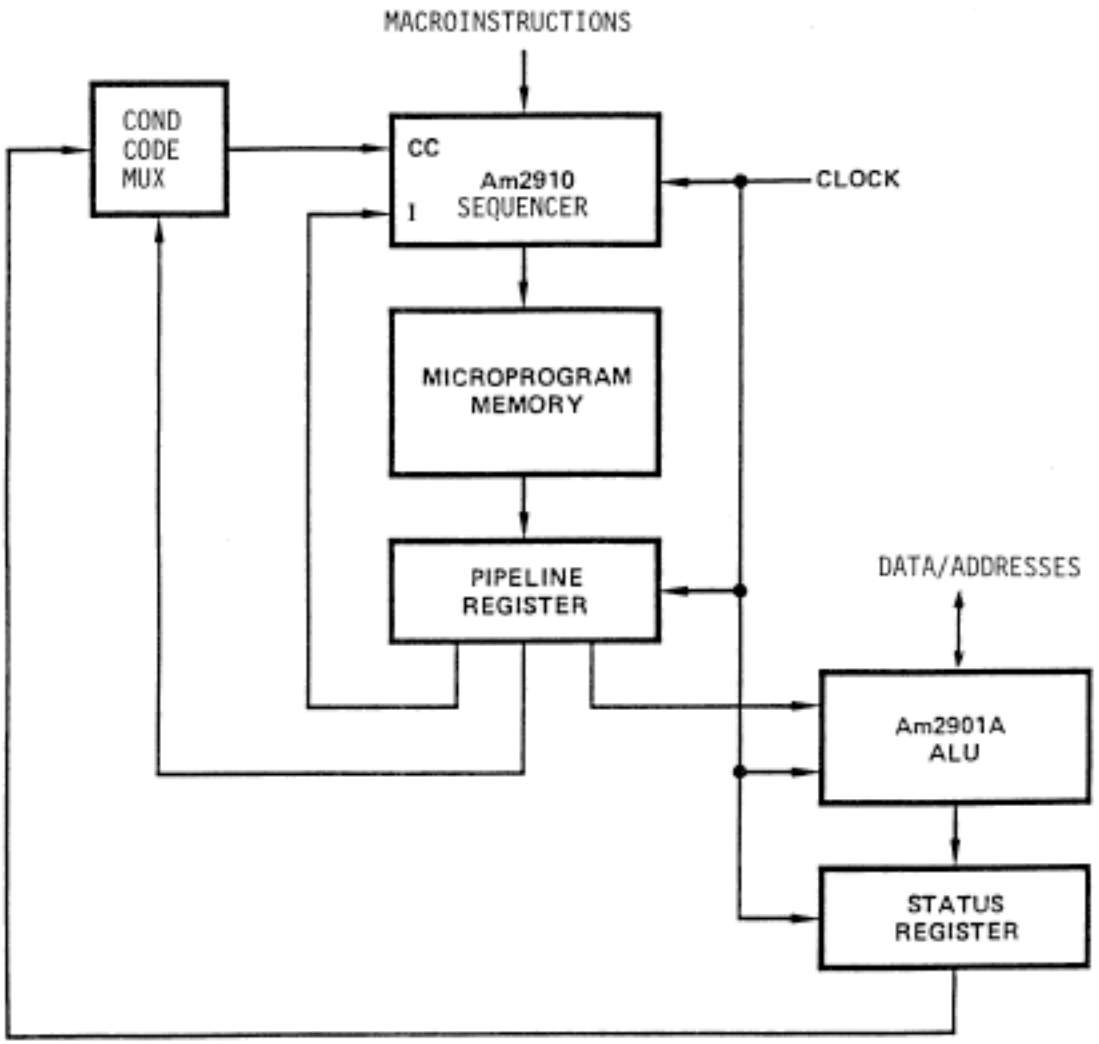
Microprogrammazione AMD 2900

HIERARCHY OF COMPUTER ALGORITHM DESCRIPTIONS/LANGUAGES

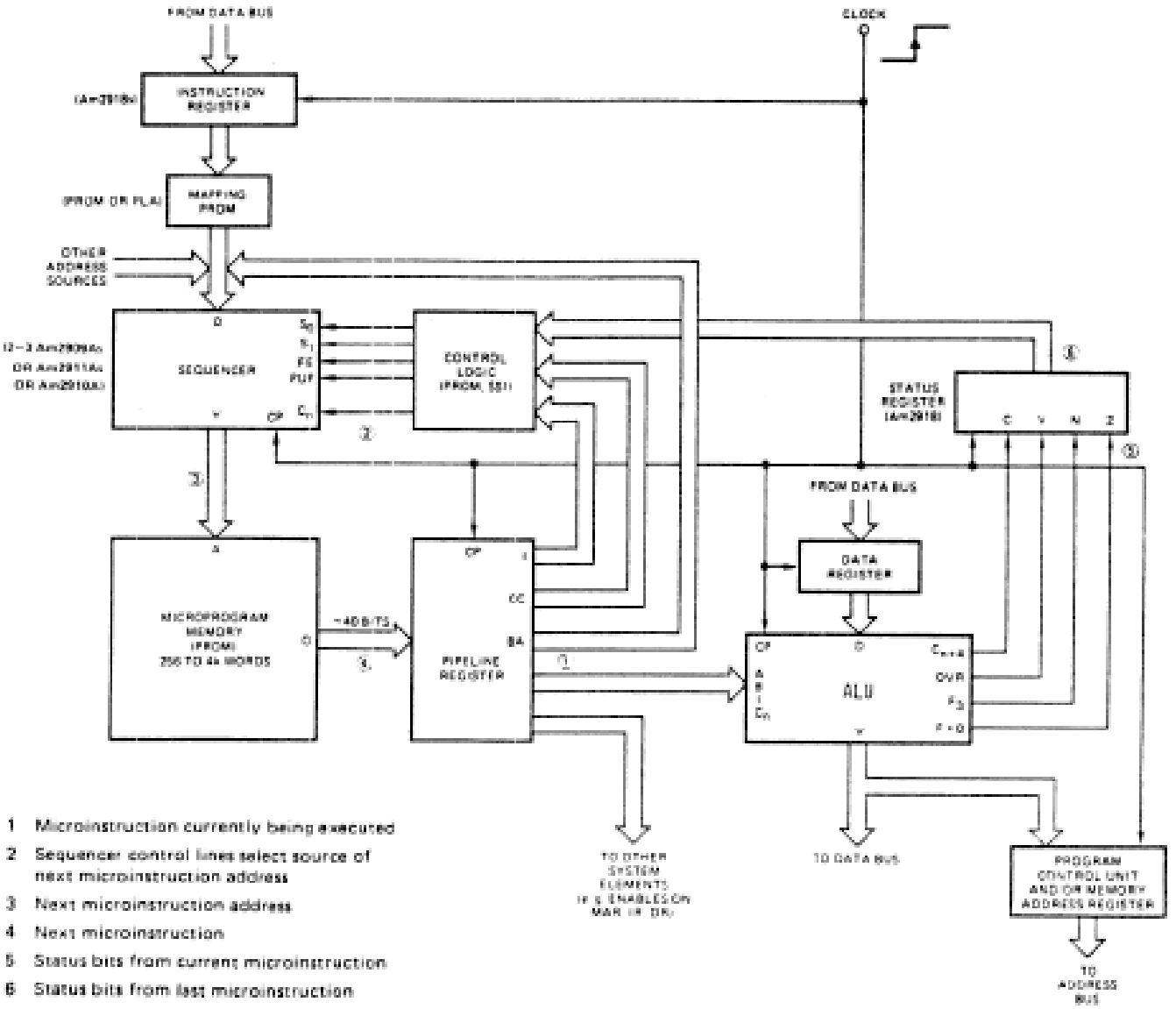
- Higher-order languages (compiler/interpreter translators)
- Lower-order languages (assembler translators)
- Machine language (macro level)
- Register-transfer languages-RTL (microprogramming)
- Boolean algebra (symbolic logic - state diagrams)
- Logic levels (timing diagrams - waveforms)

Note: One can design, implement and test algorithms on any one or more of the above levels, the choice depending upon application and constraints. Specific languages at each level are used to define a desired algorithm as well as its implementation. Various design approaches using some of the above languages are employed in this course.

GENERAL MICROPROGRAMMED ARCHITECTURE



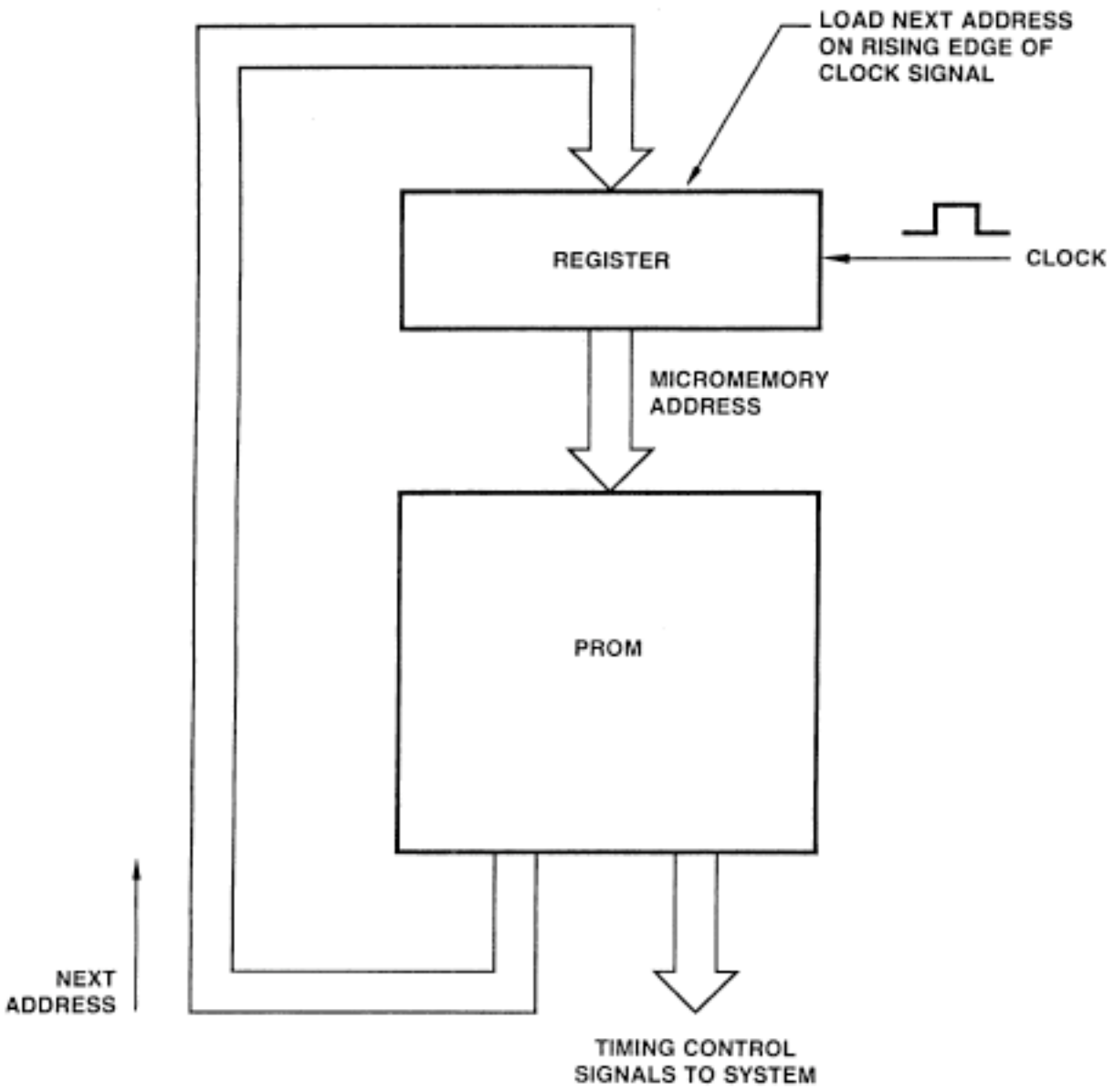
GENERAL MICROPROGRAMMED SYSTEM



1. Microinstruction currently being executed
2. Sequencer control lines select source of next microinstruction address
3. Next microinstruction address
4. Next microinstruction
5. Status bits from current microinstruction
6. Status bits from last microinstruction

THE SIMPLEST CONTROL UNIT

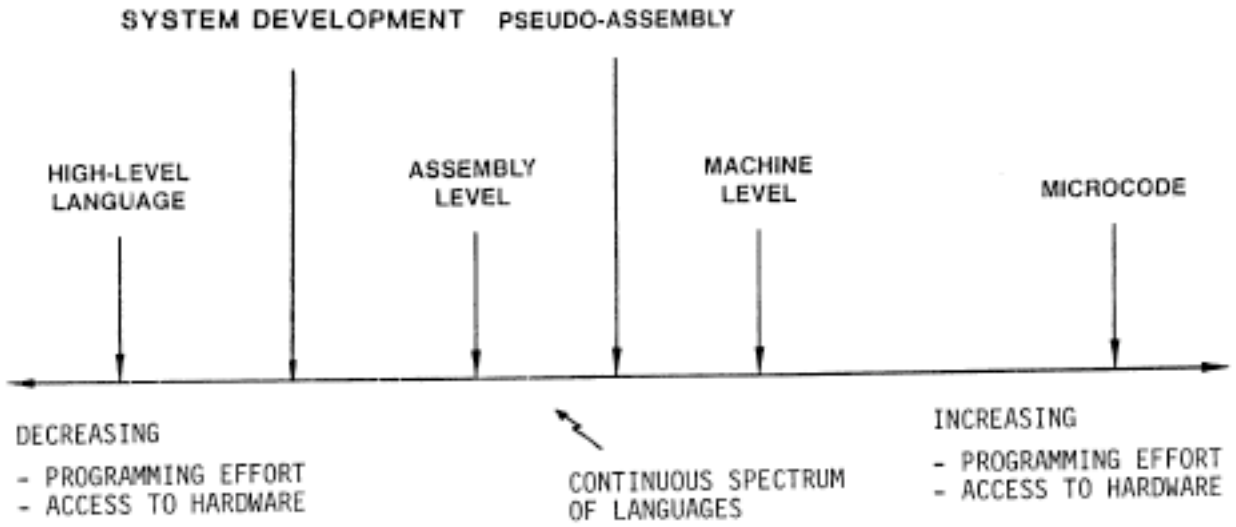
CCU - Computer Control Unit

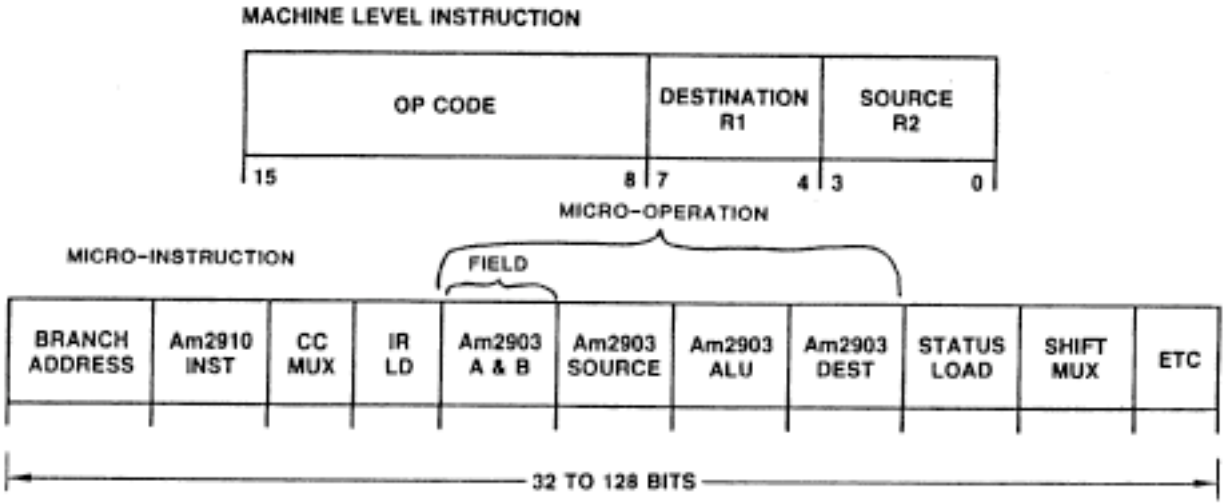


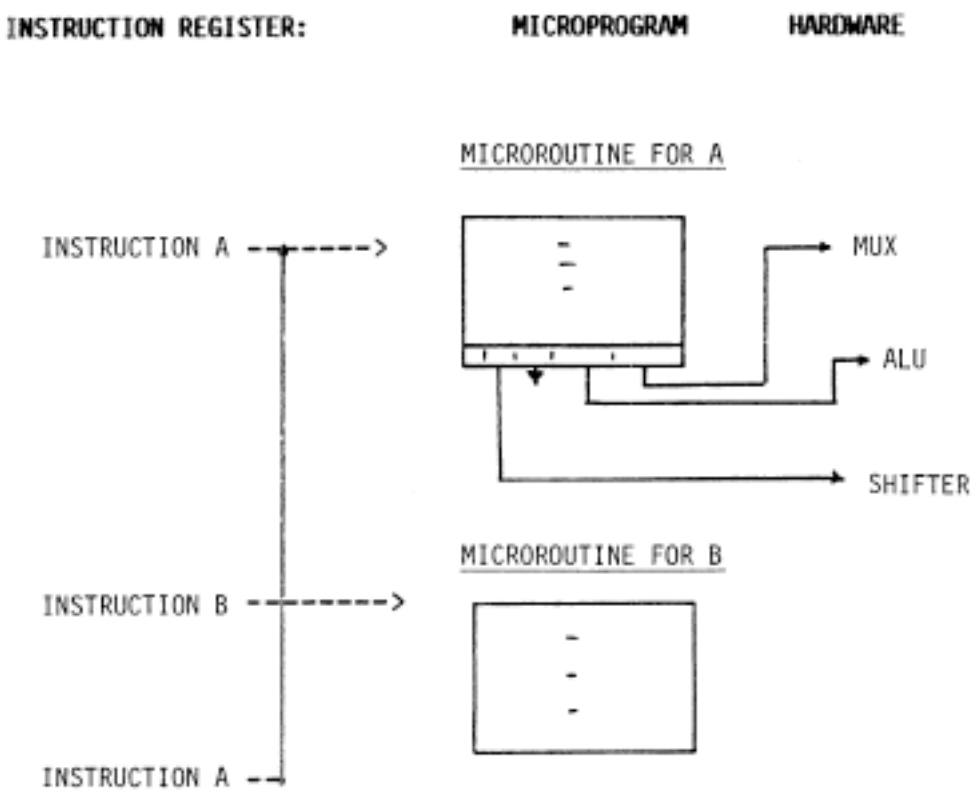
- **High Level Languages (HLL)** - Basic, FORTRAN, Pascal, ADA, etc.
  - expressed in pseudo-math (Z=X+Y)
  - converted to machine language (ML) by compiler/interpreter
  - each HLL statement translates into many ML statements
  - user is largely isolated from the particular hardware system
  - fixed instruction set (FIS)

- **Assembly Language**
  - expressed in mnemonics (ADD R1, R2)
  - converted to machine language by assembler
  - ratio to machine language statements is usually 1:1
  - user no longer isolated from knowledge of system hardware
  - fixed instruction set (operations and format)

- **Machine Language**
  - expressed in binary code (01101110)
  - each machine language instruction interpreted by a microprogram routine
  - fixed instruction set (operations and format)
  - knowledge of system hardware
- **Register Transfer Language (Microprogramming)**
  - direct control of hardware at register transfer level
  - must know complete system hardware
  - format of microprogram instruction statements defined
  - microprogramming often stored in PROM (firmware)
- **Boolean Language (Hardware logic)**
  - logic function realization in SSI/MSI circuits


SYSTEM DEVELOPMENT
PSEUDO-ASSEMBLY
HIGH-LEVEL LANGUAGE
ASSEMBLY LEVEL
MACHINE LEVEL
MICROCODE
DECREASING
- PROGRAMMING EFFORT
- ACCESS TO HARDWARE
CONTINUOUS SPECTRUM OF LANGUAGES
INCREASING
- PROGRAMMING EFFORT
- ACCESS TO HARDWARE


MACHINE LEVEL INSTRUCTION
OP CODE
DESTINATION R1
SOURCE R2
15
8
7
4
3
0
MICRO-OPERATION
MICRO-INSTRUCTION
FIELD
BRANCH ADDRESS
Am2910 INST
CC MUX
IR LD
Am2903 A & B
Am2903 SOURCE
Am2903 ALU
Am2903 DEST
STATUS LOAD
SHIFT MUX
ETC
32 TO 128 BITS



Each machine instruction causes a specific microroutine to be executed.