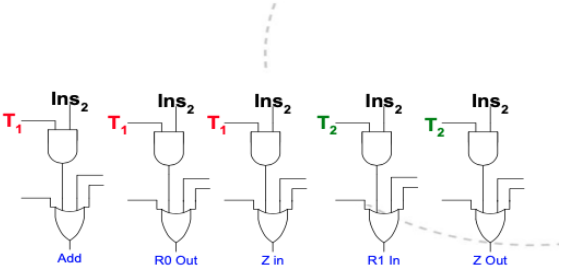
Week 12

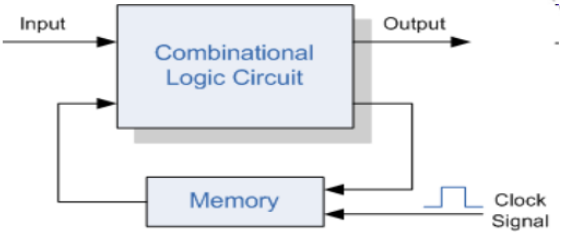
FSM

To typar kretsar

- Kombinatoriskekretsar
  - Feedforward
  - Ikkje tidssensitiv (time-independent)
  - Neste = f(input)



- Sekvensiellekretsar
  - Tilbakekopling
  - Minne
  - Neste = f(input, state)
  - Tilstandsmaskin (state machine)

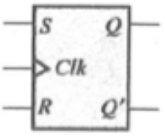
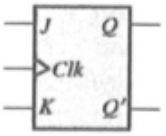
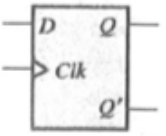
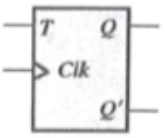


Vi har to typer kretser: kombinatoriske kretser og sekvensielle kretser.

- Kombinatoriske
  - Kretser der utgangen kun er avhengig av inngangen. Det er en funksjon
  - De er ikke tidssensitive. Er aldri noe informasjon is systemet hva utgangen var tidligere.

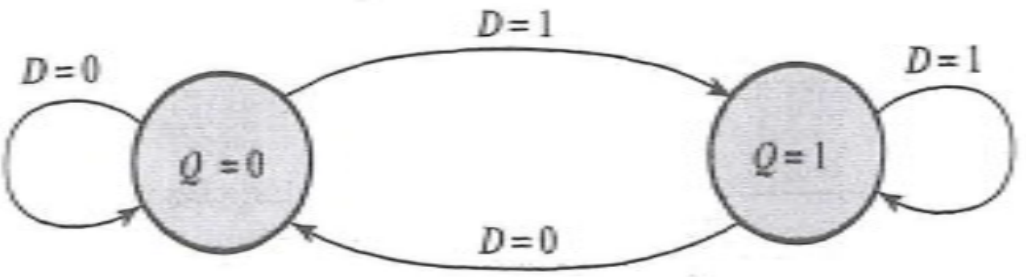
- Sekvensielle
  - Har minne så de vet hva de tidligere utgansverdiene er
  - I diagrammet over ser vi at den sammenstår av kombinatorisk krets. Utgangen er avhengig av inngangen og den forrige utgangen (blir lagret i memory)
  - Dette går sekvensielt med klokken.

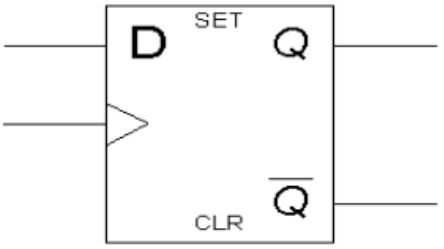
Fudamentalt : Flip-Flop

FLIP-FLOP NAME	FLIP-FLOP SYMBOL	CHARACTERISTIC TABLE	CHARACTERISTIC EQUATION	EXCITATION TABLE
SR		S R | Q(next) 0 0 | Q 0 1 | 0 1 0 | 1 1 1 | NA	$Q(next) = S + R'Q$ $SR = 0$	Q Q(next) | S R 0 0 | 0 X 0 1 | 1 0 1 0 | 0 1 1 1 | X 0
JK		J K | Q(next) 0 0 | Q 0 1 | 0 1 0 | 1 1 1 | Q′	$Q(next) = JQ' + K'Q$	Q Q(next) | J K 0 0 | 0 X 0 1 | 1 X 1 0 | X 1 1 1 | X 0
D		D | Q(next) 0 | 0 1 | 1	$Q(next) = D$	Q Q(next) | D 0 0 | 0 0 1 | 1 1 0 | 0 1 1 | 1
T		T | Q(next) 0 | Q 1 | Q′	$Q(next) = TQ' + T'Q$	Q Q(next) | T 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

I boken Geiski så står det mye om kretser. Vi ønsker å vite hvordan ting genrelt fungerer og ikke nødvendigvis de små spesifikasjonene på hvorfor.
Vi kommer til å bruke kun et element => flip-flop. Denne har en klokkeinngang. Den er aktiv når klokken går opp.

D



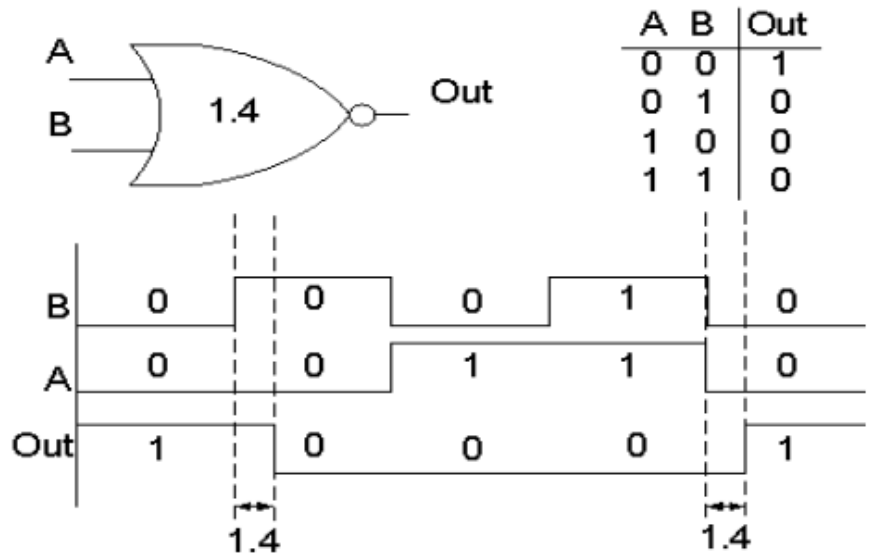


D	CLK	Q(next)	D Q(next)
0	0	NC	NC
1	0	NC	NC
0	1	NC	NC
1	1	NC	NC
0	↑	0	1
1	↑	1	0

NC = No Change

Vi kan lage et tilstandsdiagram for en flip-flop. Denne beskriver hvilken tilstand flip-floppen er i basert på hva vi gjør med den.

En endret verdi her betyr hvilken verdi har Q.



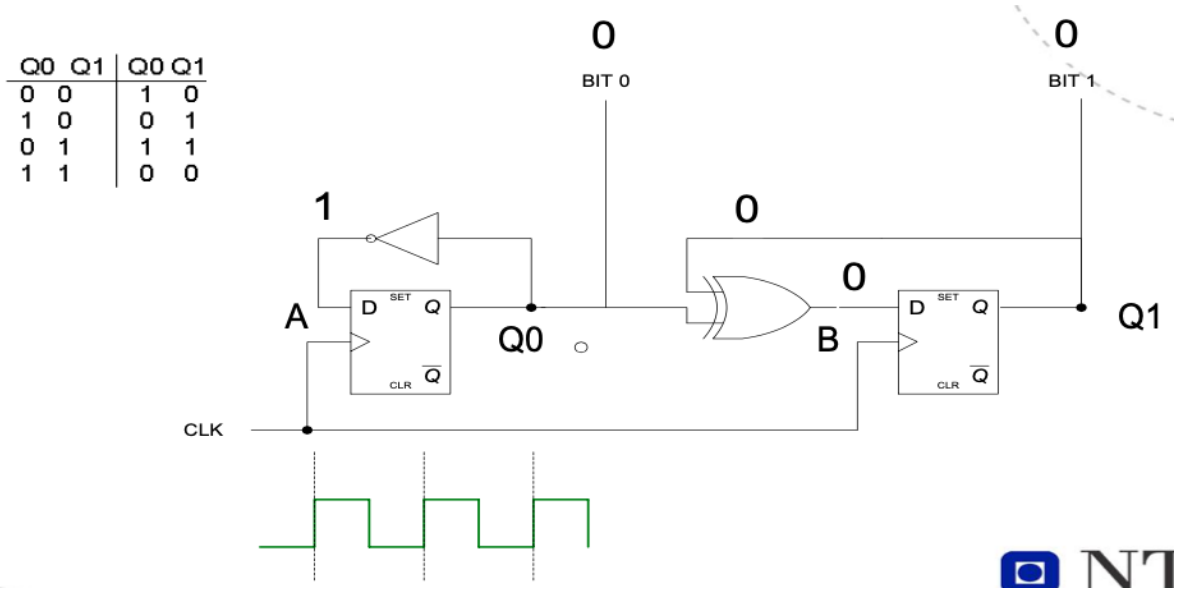
NOR-port

- Har forsinkelse
- Her 1.4 (f.eks ns)
- Out skiftar litt seinare (1.4)

I sekvensiell logikk er det tilbakekopling

- Kan gi ustabilitet eller uønskja oppførsel

Før vi ser på sekvensielle kretser så er det viktig å huske på at kretser har en delay. Dersom vi har en NOR-port og vi får inn A og B, så vil det f.eks ta 1.4 ns før vi får den riktige outputen. Dette kan gjøre sekvensiell logikk, som har tilbakekobling, litt ustabil.



Her har vi en teller som teller to bit. Vi har en tilbakekobling på Q0 og Q1.

La oss si at Q0 = 0 og Q1 = 0: Vi har en inverter for Q0 så vi får inn 1 på A. Vi har en 0 på Q1. Denne går tilbake til XOR-porten samtidig som vi da har Q0=0 (den inverterte Q0 har ikke kommet til XOR enda) siden vi har to innganger på 0 i XOR så vil vi få 0 i B.
Nå kommer klokkepulsen. Vi oppdaterer Q0 = 1, tidligere B verdi er 0, dermed vil XOR være 1.
Neste klokkepuls vil føre til at Q1 = 1 og Q0 = 0 på grunn av den forrige tilbakekoblingen.

Vi har en tabell øverst som viser hvordan denne telleren shifter for hver klokkefrekvens.

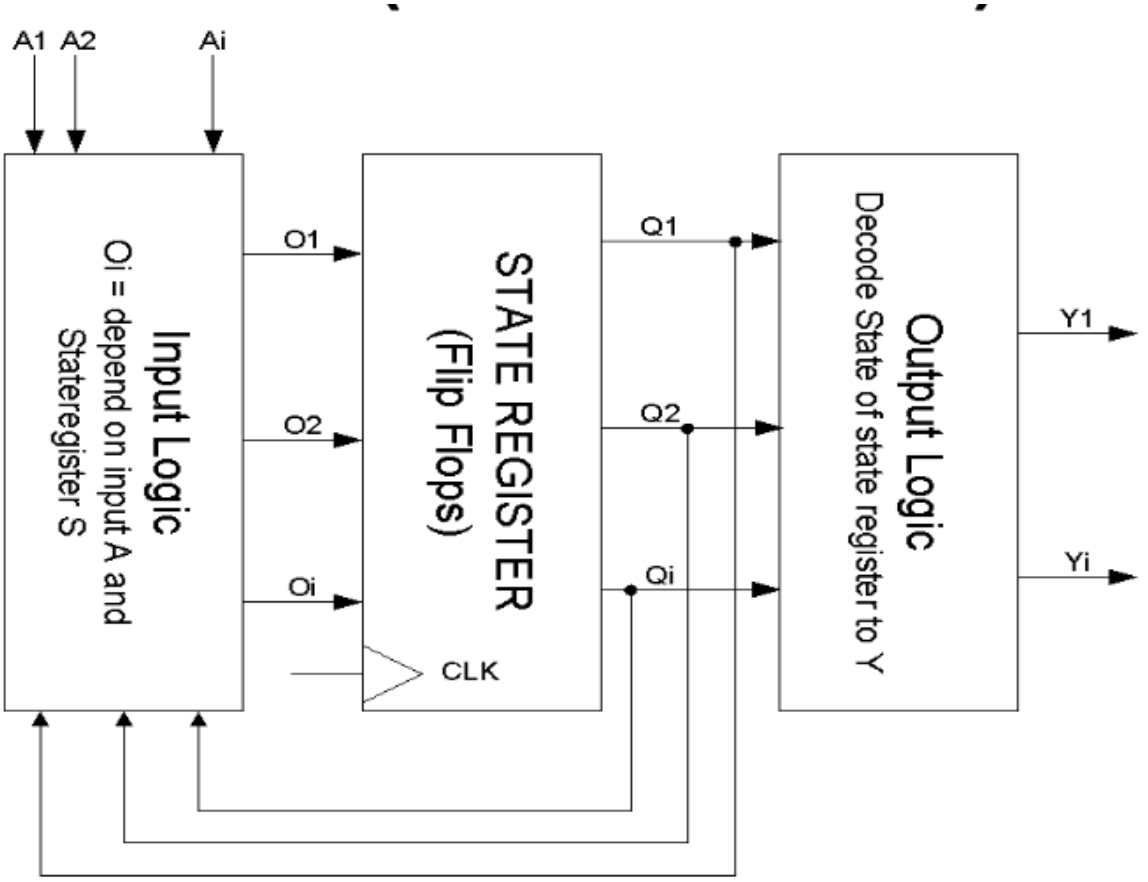
Forsinkelse er essensielt her. Forsinkelsen gjennom inverteren og XOR-porten må være mindre enn klokkeperioden!

FSM Moore og Mealy

Vi har to Finite State Machines (FSM)

- State-based **Moore machine**
  - Endrer utgang kun når tilstandens registre endrer seg
- Input-base **Mealy machine**
  - Utgang kan endre seg sammen med inngangssignal uavhengig av klokken

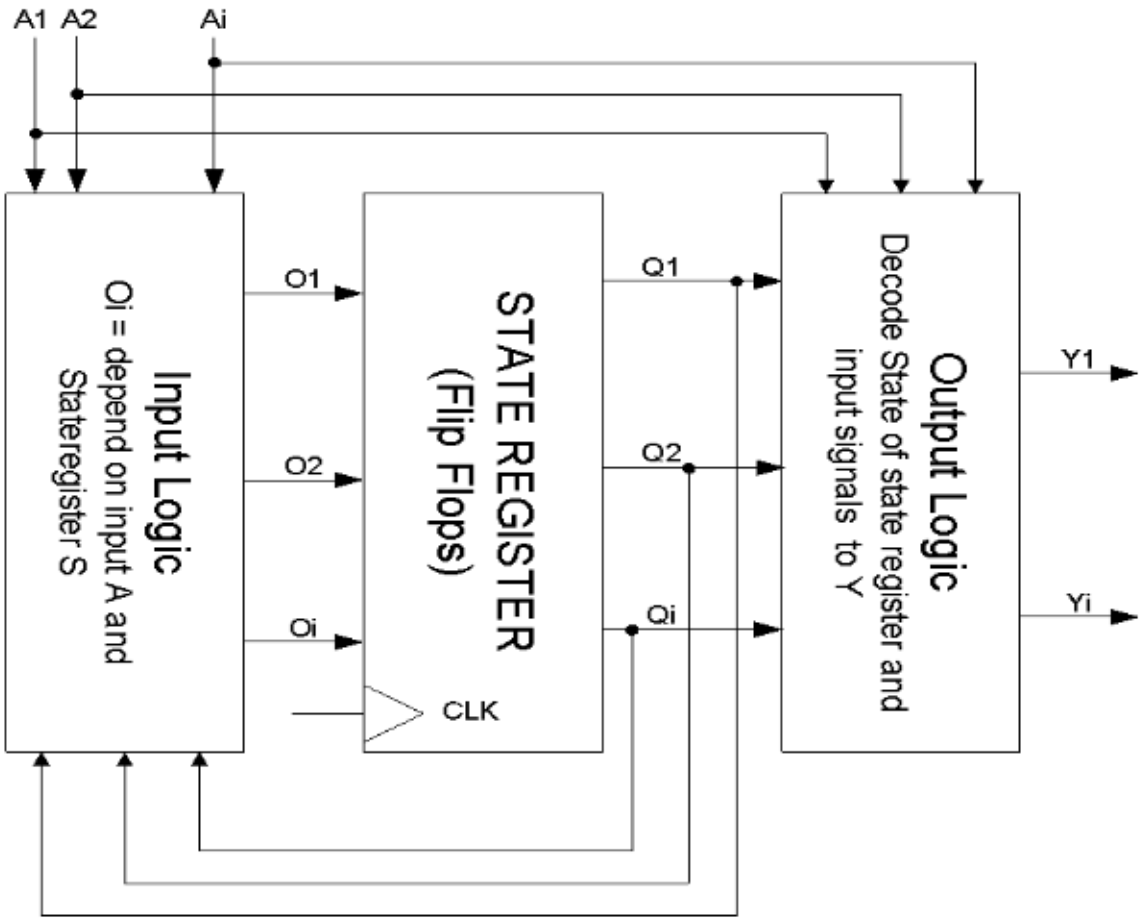
Moore FSM



Her er en Moore FSM. Til venstre har vi noe logikk som forteller oss hva registeret skal oppdateres på. Denne logikken har eksterne signal (A1, A2, Ai) og en tilbakekobling fra tilstanden STATE.

I dette eksempele har STATE REGISTER tre utganger. Dette betyr at vi har 2^3 mulige tilstander. Vi kan putte på logikk på utgangen, som vi har gjort her, for å få kun to utgangssignal (Y1 og Y2).

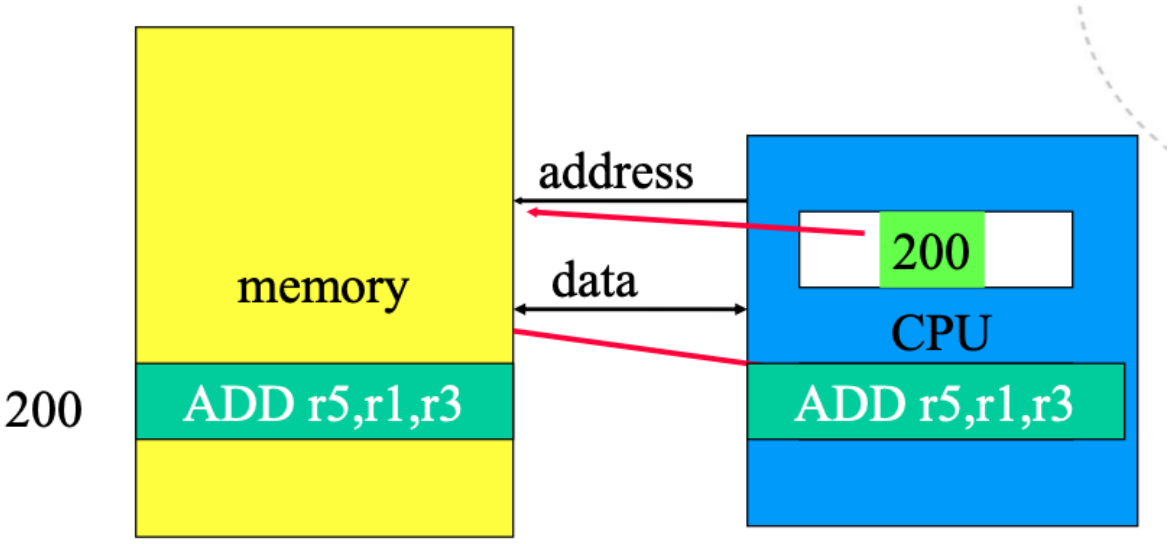
Mealy FSM



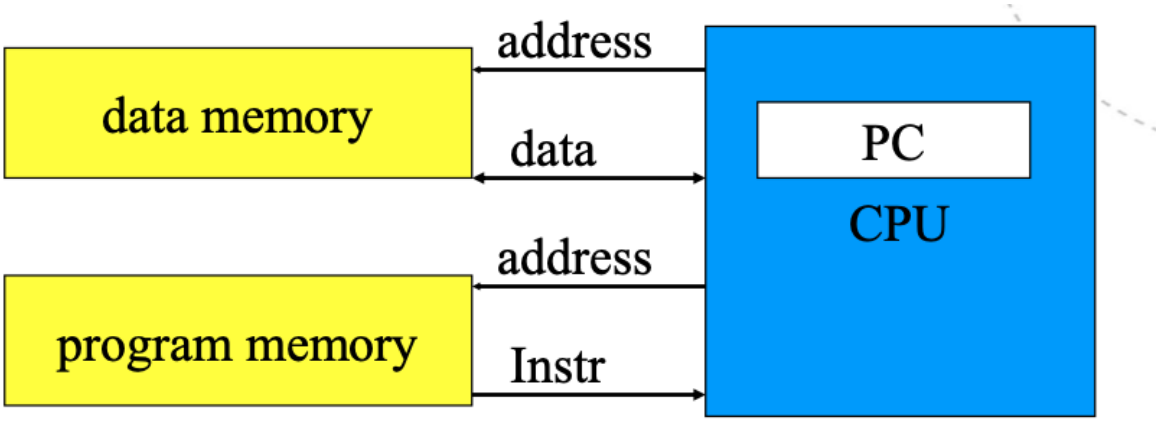
Forskjellen på Mealy og Moore er at Mealy bruker de eksterne signalene i output logikken sammen med Q1, Q2 og Q3.

ISA

Adresserom



von Neuman architecture computer



Harvard architecture

Harvard arkitektur er noe liknende IJVM bruker. Vi kan behandle adresse og data samtidig i Harvard arkitekturen.

Instruksjonssett

- Komplekst/enkelt
  - Type instruksjoner
  - Lengde
- Påvirkning
  - Vi får stor kontrollenhet for å håndtere mange komplekse instruksjoner
  - Vi får en lang lengste-path. Klokkefrekvensen må være lengre.
  - Stort areal

CISC (Complex Instruction Set Computer)

- Avanserte instruksjonar (kan utføre mykje)
  - Bruke fleire einheitar
  - Nærare høgnivå språk
- Fleksibelt instruksjonsett (Viss mikroprogram kan oppdaterast/endrast)
- Variabel lengde på instruksjonar
- Variabel tid på å utføre instruksjonar
- Mikroprogram (ikkje alltid)
- Mange typar instruksjonar og adresseringsmåtar

ADD MinneSvar, MinneAdrX, MinneAdr,Y

RISC (Reduced Instruction Set Computer)

- Enkle instruksjonar
  - Register til register instruksjonar
  - Bruke mange enkle instruksjonar på å utføre oppgåver
- Fast instruksjonsett (Hardwired, digitallogikk (FSM))
- Fast lengde på instruksjonar
- Fast tid på å utføre instruksjonar
- Ei maskinvare statemaskin som styreeinheit (eller helst berre logikk)
- Kunn ein type adresering LOAD/STORE
- Rask

Load R0, R1

Store R0, R1

ADD R0, R1, R2

Maskin typer

Ein-sykel maskin

- Ein Instruksjon kvar klokkeperiode
- Instruksjonar må være enkle
  - Copy R1 <-R2
  - Add R1, R2, R3
  - Load R1, MAR
  - Store R1, MAR

Fleir-sykel maskin

- Fleire klokkeperiodar/sykel
- «Komplekse» instruksjonar
  - Copy R1 <- minneAdr (mAdr)
  - Add mAdr, mAdr, mAdr
  - Load R1, mAdr + R2 (offset)
  - Store R1, mAdr + R2 (offset)