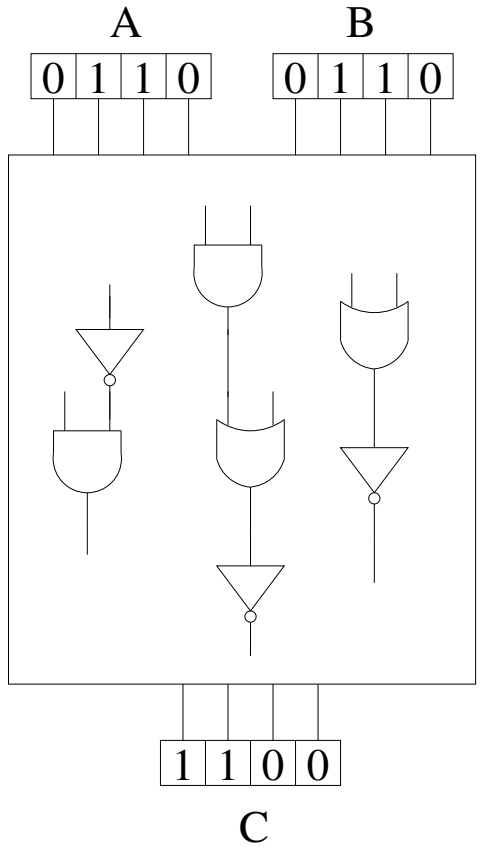
Sist

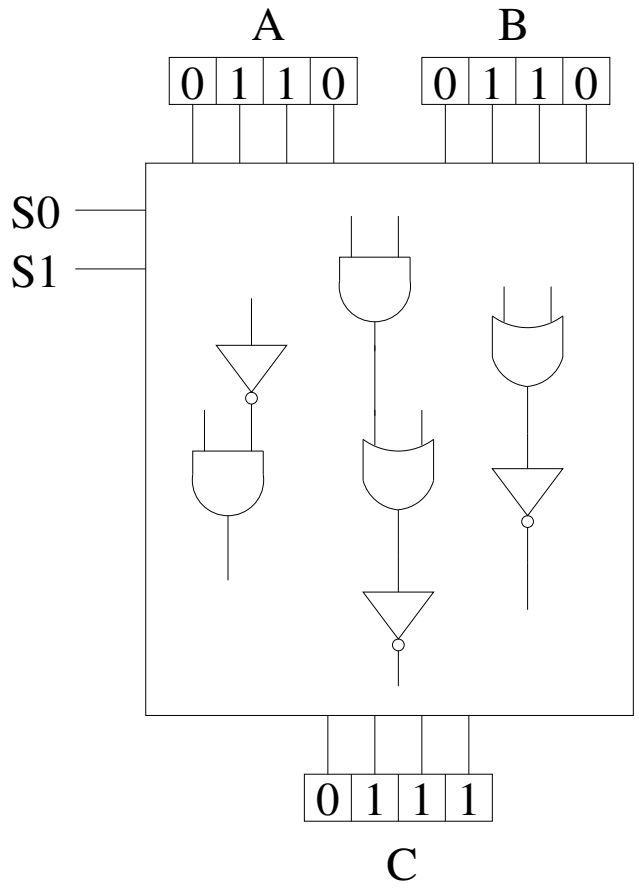
- Transistorer
- Porter
- Sannhetstabell
- Skrive ned krets med AND, OR og NOT-porter
- Forenkle kretsen med Boolsk algebra
- Tegne logisk krets
- Logisk krets som adderer tall

Adderings-krets



C = A + B (12 = 6 +6). Logiske kretser med AND, OR og NOT er konstruert inne i boksen på en slik måte at de alltid regner ut riktig resultat. Kretsen leser fra og skriver til 4-bits registere.

Arithmetic Logic Unit (ALU)



En ALU med fire funksjoner. C = A + 1 (7 = 6 + 1). Her styrer kontroll-bitene S0 og S1 kretsen til å øke A med en og legge resultatet i C.

ALU-operasjoner

- Adder
- Subtraher
- Inkrement ($++$)
- Dekrement ($--$)
- Multipliser
- Divider
- Shift (flytt alle bit i en retning)
- Sammenligne
- AND, OR, NOT, XOR

Ved å sette riktig verdi på noen kontroll-bit som sendes til ALU, velger man ønsket funksjon.

Tidlig ALU-design

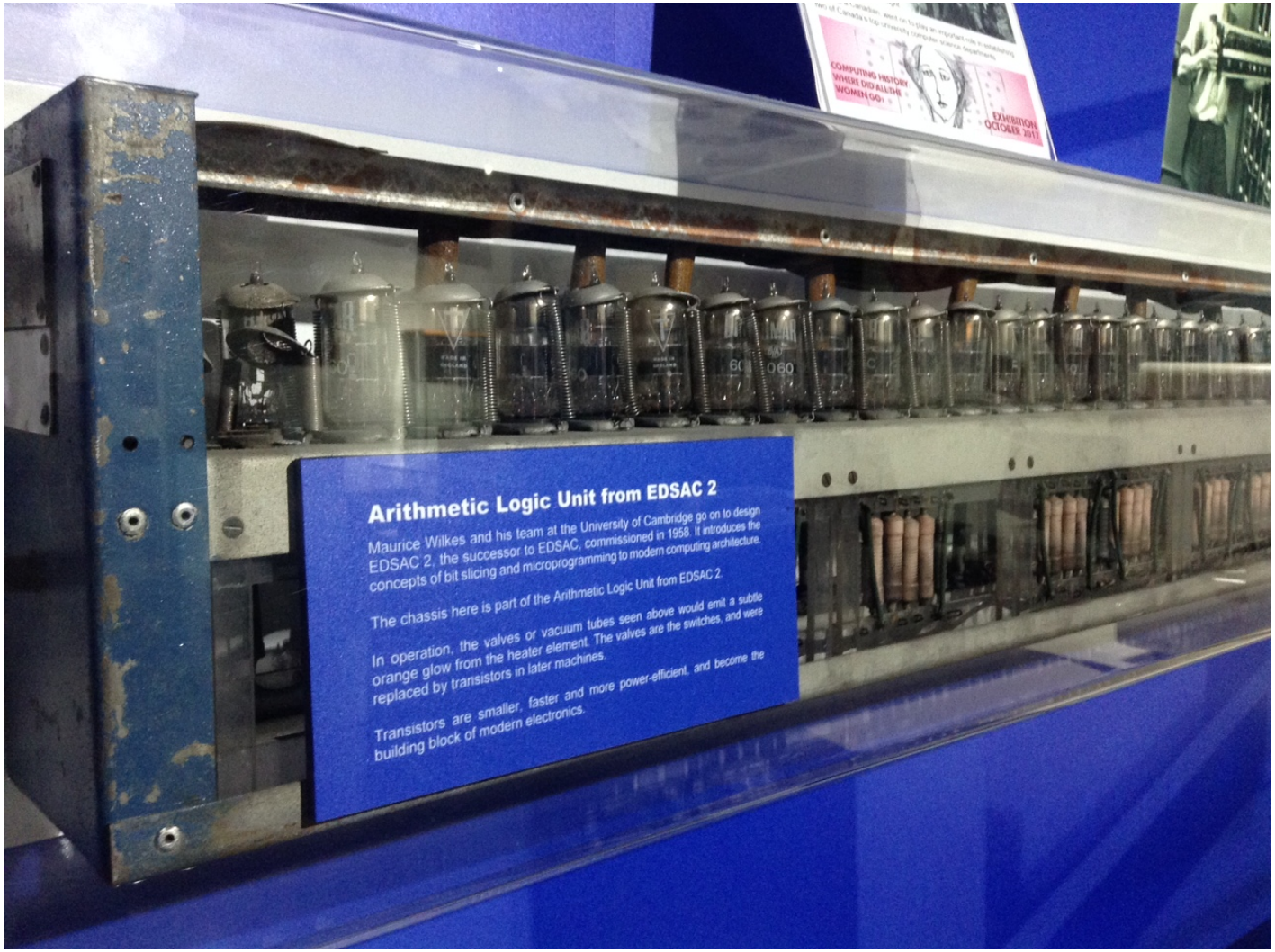
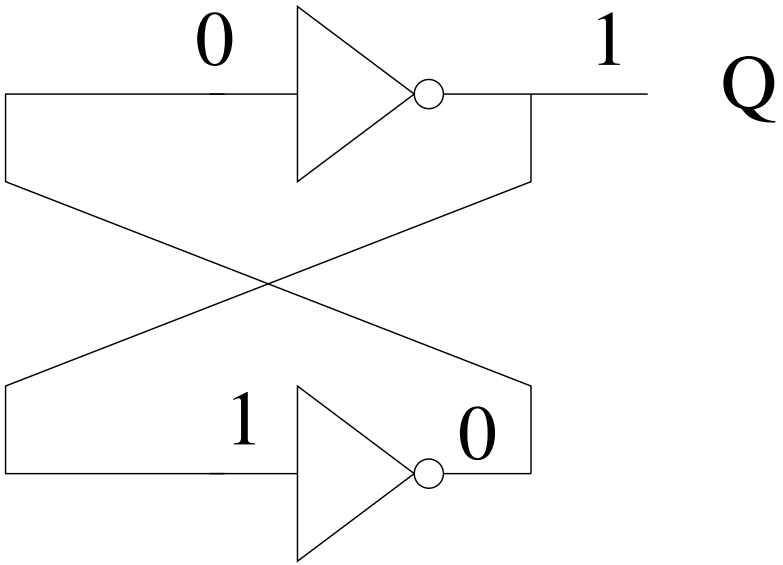


Figure: ALU-en til EDSAC 2 fra 1958 hadde radiorør, som senere ble erstattet av transistorer.

Vipper og registere

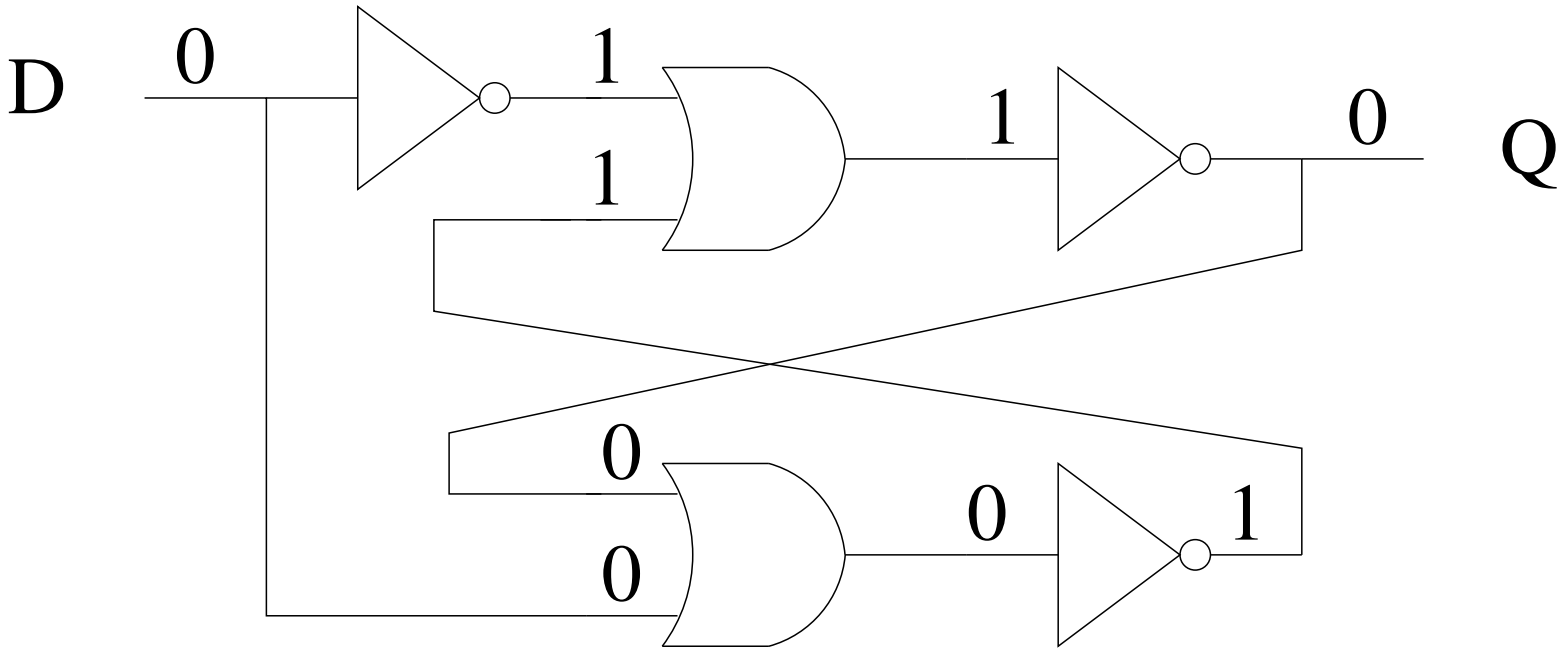
- Kondensatorer kan lagre ladning (0 og 1) men er ikke hurtig nok. Denne teknokogien brukes i RAM.
- Av porter kan man lage en enhet for å lagre nuller og enere: vippe (flip-flop)
- Ekstremt hurtig, mye raskere enn RAM
- Et 64 bit register består av 64 vipper
- For å lagre noe med porter, må vi lage en lukket krets slik at bit-verdiene bevares.

Lagre en bit



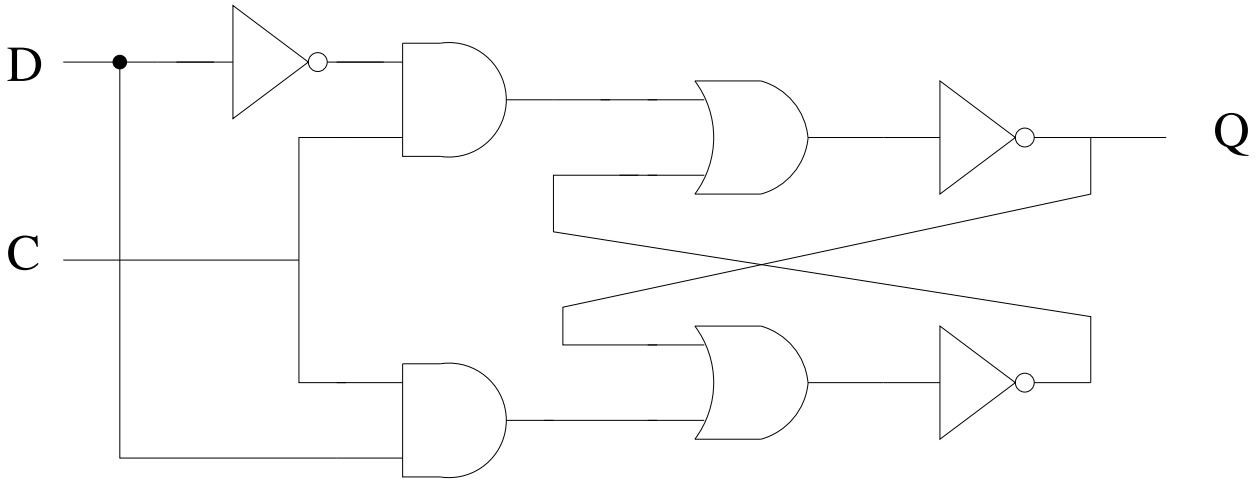
Lagring av verdien Q = 1. Hva er problemet?

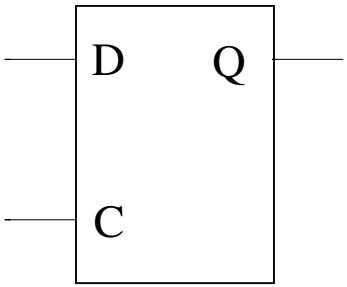
Endre bit-verdi



Endring av verdien Q til 0. Hva er problemet nå?

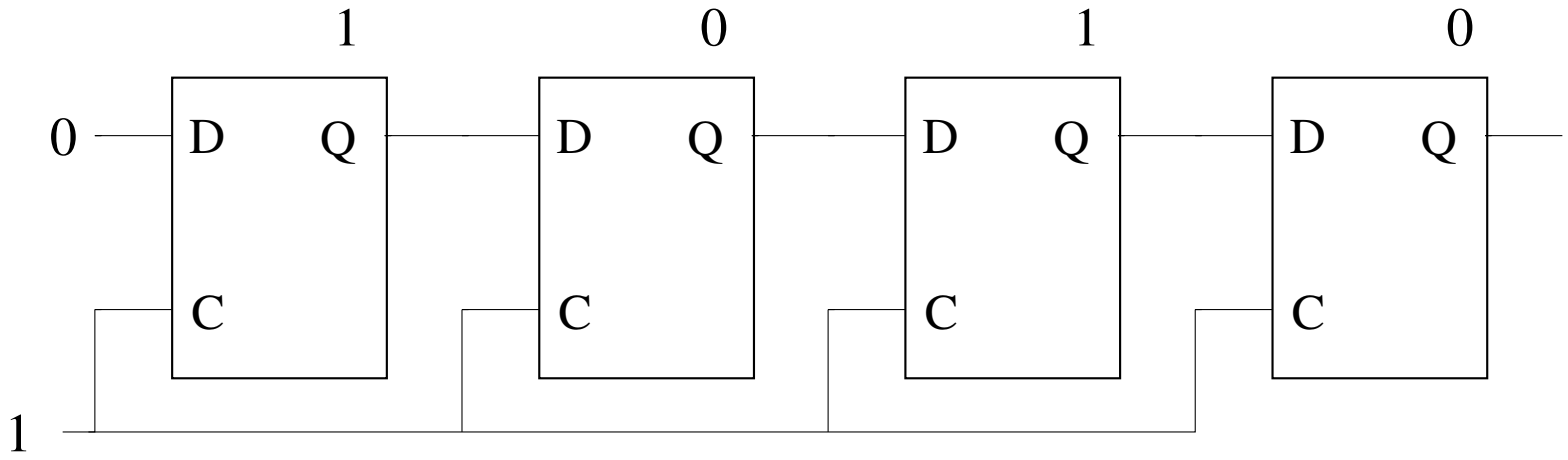
D-lås (D-latch)





- C = 1 $\longrightarrow$ verdien inn fra D lagres
- C = 0 $\longrightarrow$ den lagrede verdien beholdes, uavhengig av verdien inn fra D

Shift-register



Fire D-låser settes sammen til et shift-register. Vi ønsker å kunne utføre en operasjon som 1010 -> 0101. Hva er problemet?

Simulering av shift-register

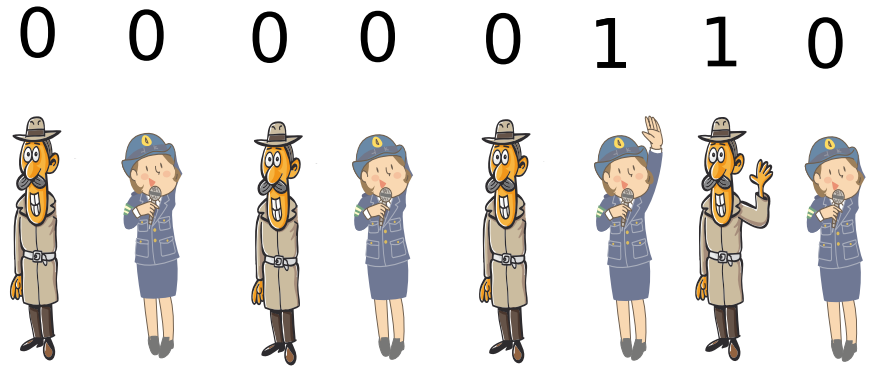


Figure: Åtte menneskelige D-låser på rad. Armen opp betyr 1, ned betyr 0. Hver D-lås leser verdien til D-låsen til høyre for seg.

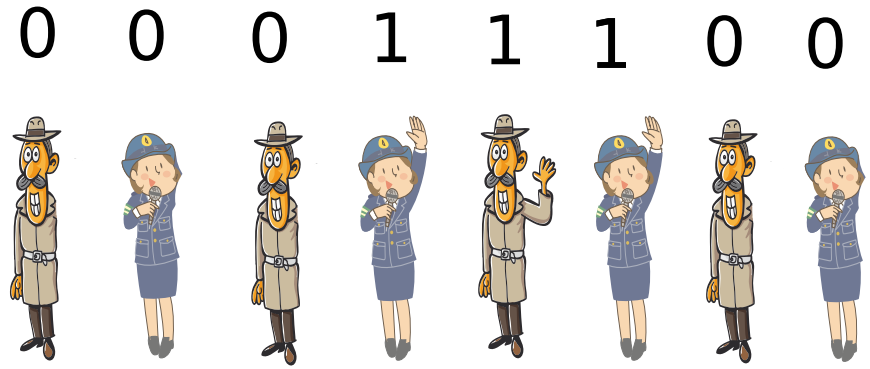
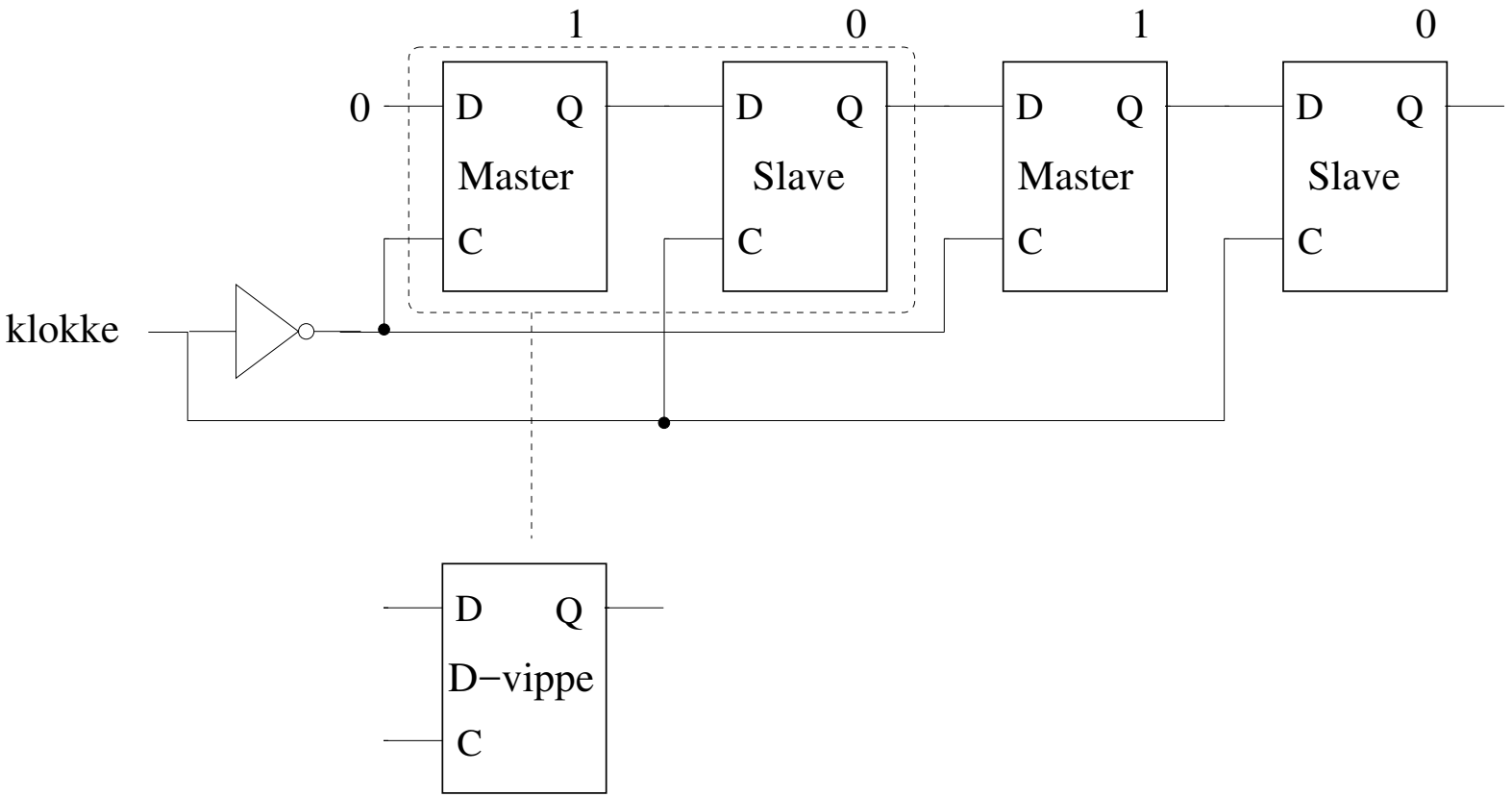


Figure: De menneskelige D-låsene reagerer ikke nøyaktig like fort og dermed kan dette bli neste tilstand for arrayet av bits, der en ener for mye har dukket opp.

D-Vippe



To master og slave D-låser blir til en D-vippe. Klokken går av og på og styrer når de endelige dataene lagres av slaven. Dette er selve CPU-klokken, som typisk har en frekvens på 1-3 GHz.

CPU-klokke

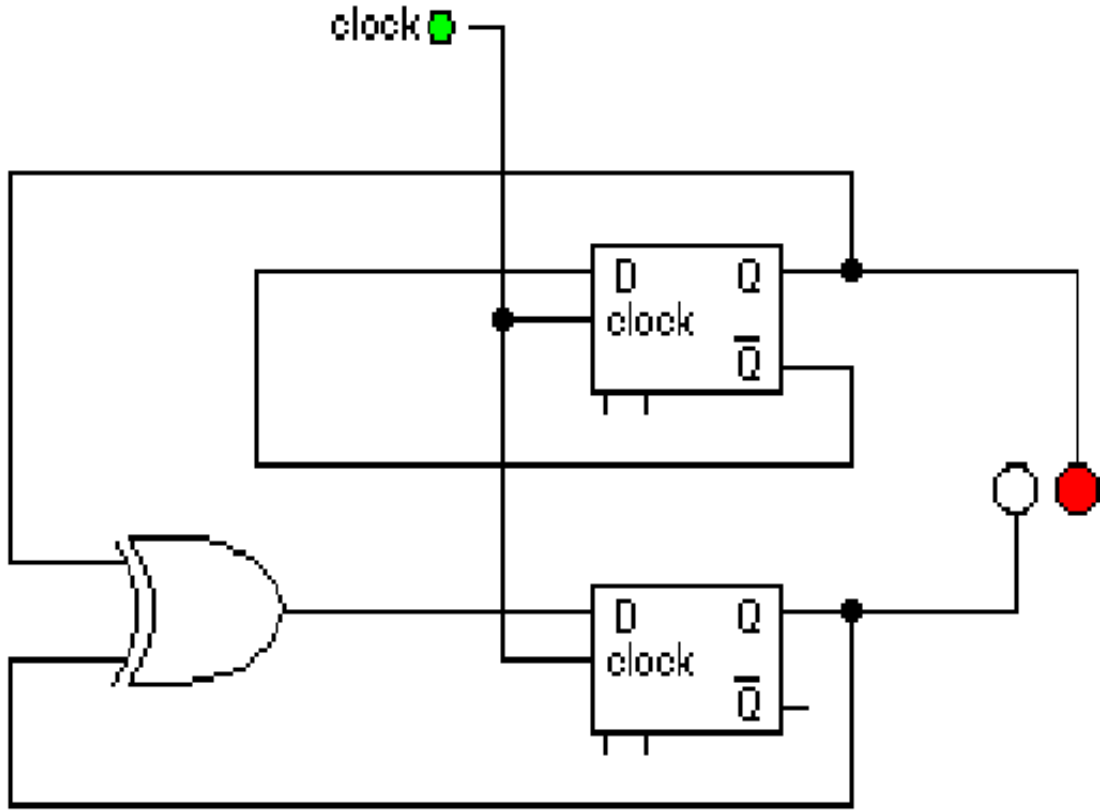
Tiden deles inn i små klokke-tikk av CPU-klokken og innenfor et slikt tikk må

- Når klokken sender en 0: alle beregninger ferdigstilles ved å strømme igjennom kretsene som adderer eller gjør annen logikk, sluttresultatet lagres hos master. Det må være ferdig før klokken switcher til 1.
- Når klokken sender en 1: slaven leser verdien fra master og lagrer den. Den begynner straks å sende dette resultatet, som er det gjeldende resultatet, ut i kretsene som er koblet til utgangen for nye beregninger.

CPU-klokken er helt essensiell for å synkronisere dataene, for hvert tikk av klokken kan et nytt sett av bergninger gjøres, for eksempel å utføre en maskin-instruksjon.

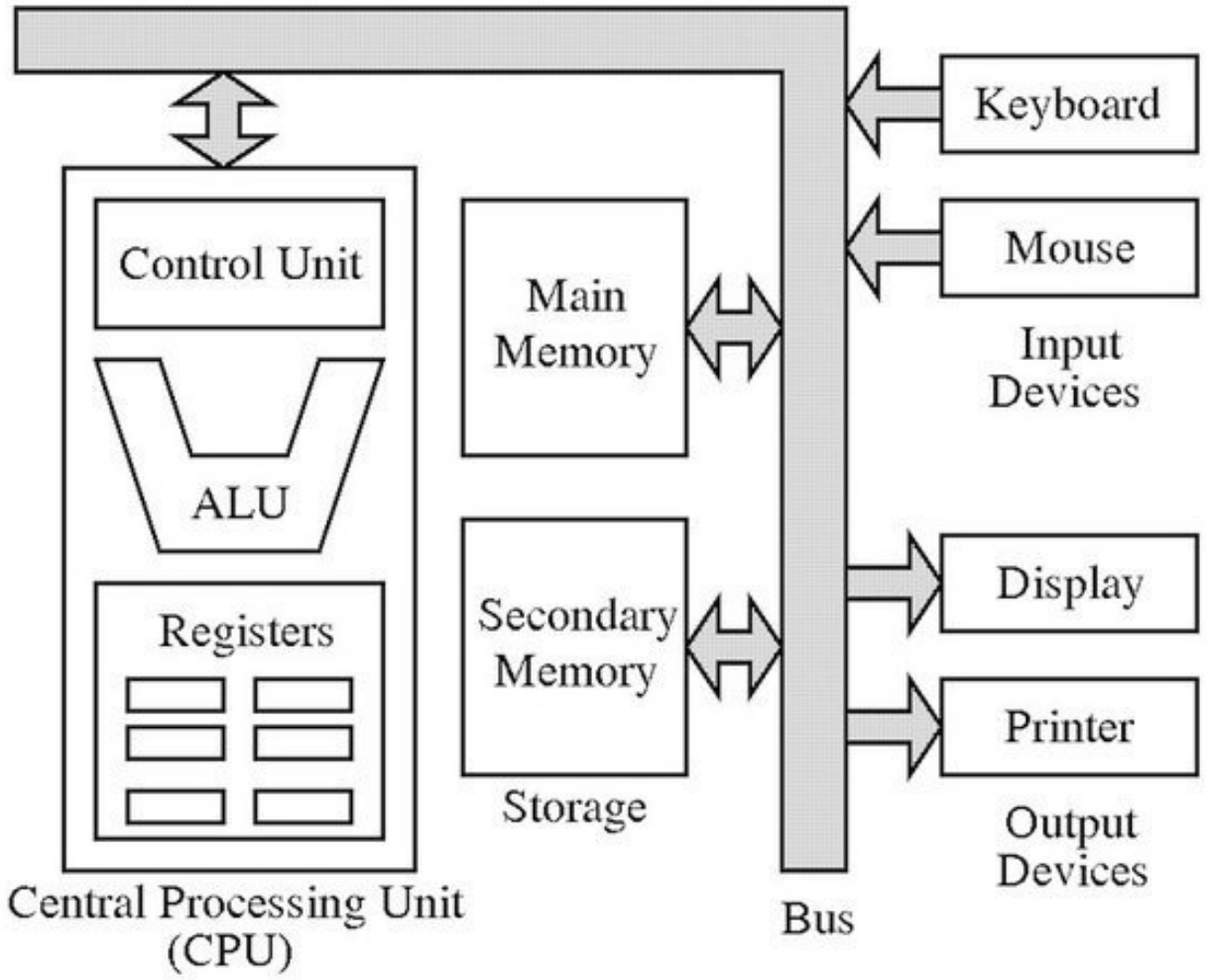
Tellere

For å kunne løpe gjennom instruksjoner i et program, trenger man en teller som kan telle oppover for hvert klokke-tikk.



En 2-bits teller som kan telle fra 0 til 3. For å lage større tall, trenger vi bare flere vipper.

Von Neumann arkitektur



Den vanligste datamaskinarkitekturen som brukes, definert av matematikk-professoren John von Neumann i 1945.

Von Neumann arkitektur

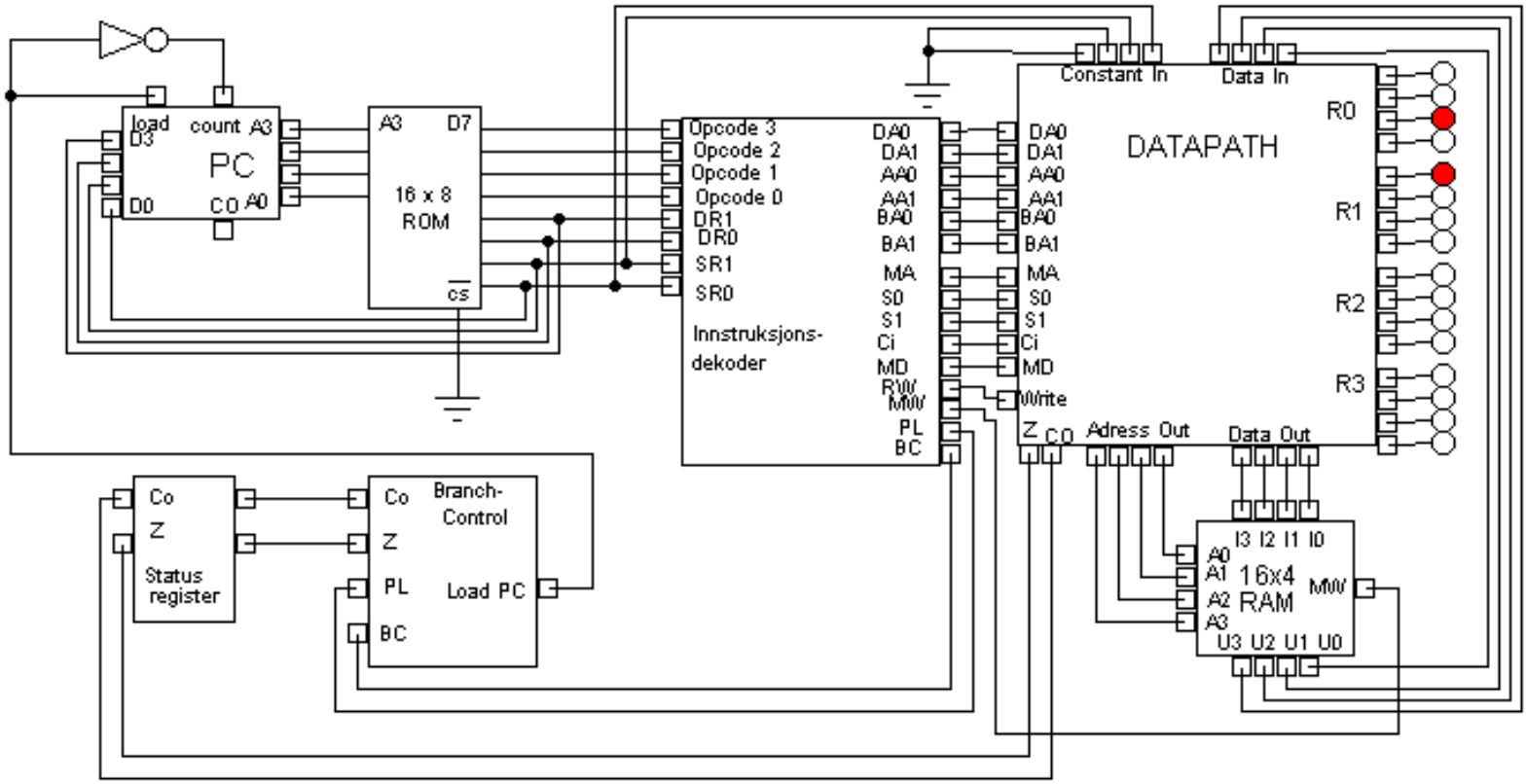
- Et arbeidsminne (internminne/RAM) som inneholder både instruksjoner og kode.
- En aritmetisk/logisk enhet (ALU - Arithmetic Logic Unit) som kan utføre matematiske og logiske operasjoner.
- En kontrollenhet som henter inn instruksjoner fra RAM, dekoder dem og sender signaler som gjør at instruksjonen blir utført.
- Registere, internlager for både instruksjoner og data inne i CPU'en.
- Enheter for input og output som gjør at CPU kan kommunisere med harddisk, tastatur, nettverk, etc.

Beregningsenheter

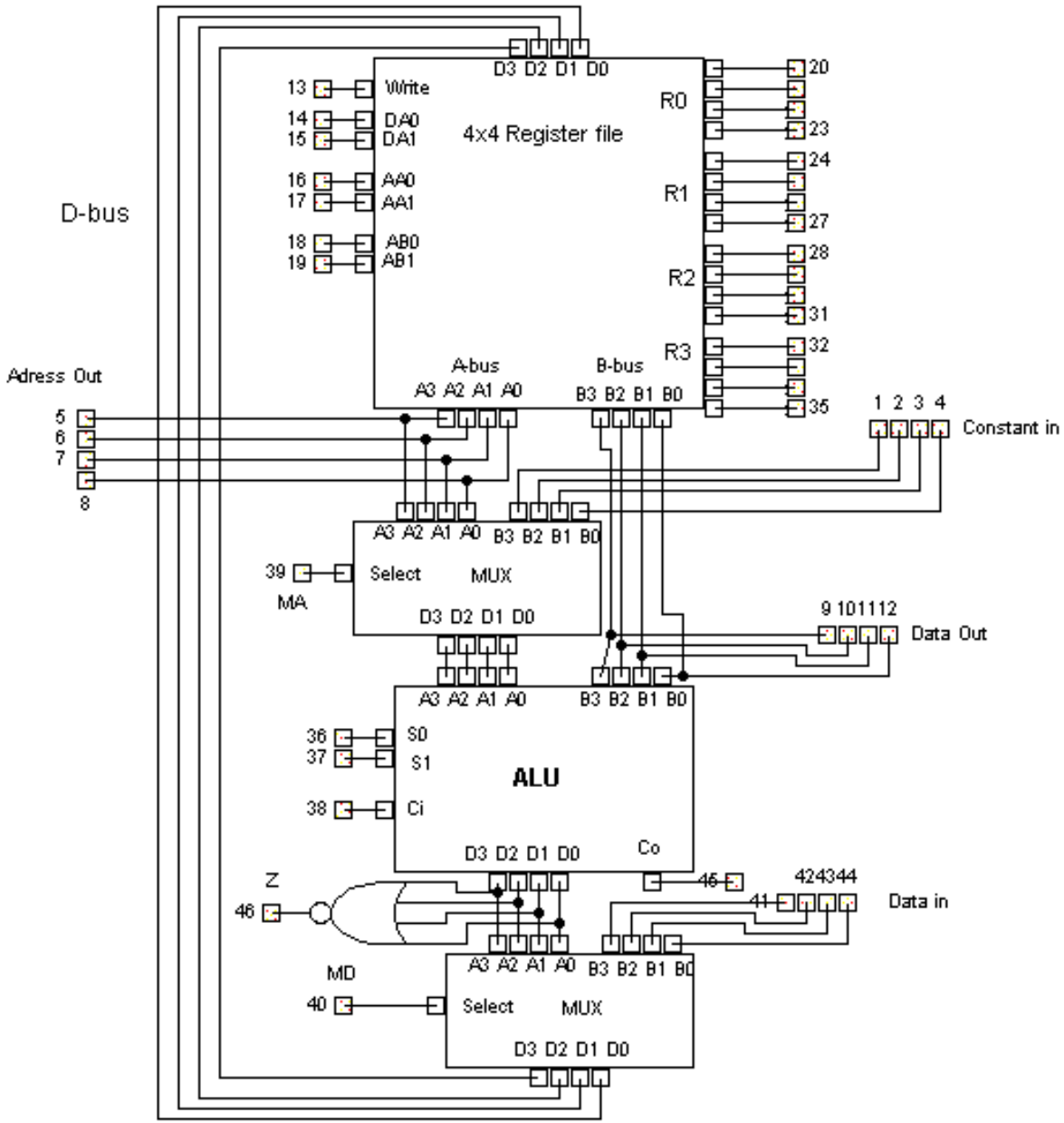
- ALU (Arithmetic Logic Unit) CPU'ens hjerne
- CPU (Central Processing Unit)
- FPU (Floating-Point Unit) vanligvis integrert i CPU
- GPU (Graphics Processing Unit) tusenvis av cores
- FPGA (Field-programmable Gate Array) programmerbar logikk
- ASIC (Application-Specific Integrated Circuit)

Simulering av CPU

Arkitekturen til en komplett 4-bit CPU som kan utføre programmer skrevet i maskinkode.



Datapath



Høynivåkode

Høynivåkoden

```
S = 0;
for(i=1;i < 4;i++)
{
   S = S + i;
}
```

betyr at en programmerer ønsker CPU'en skal utføre følgende:

```
i = 1: S = 0 + 1 = 1
i = 2: S = 1 + 2 = 3
i = 3: S = 3 + 3 = 6
```

instruksjoner

Noen av maskininstruksjonene til den simulerte CPUen er definert slik

binært Nr	operand1	operand2	Nr	Navn
0010	DR	tall	2	MOVI
0100	DR	SR	4	ADD
1100	DR	SR	12	CMP
1111	nr	nr	15	JNE

Dette er med på å definere denne datamaskinarkitekturen.

Assemblykode

Assemblykode for for-løkken:

```
0 MOVI R0 <- 3            (MOV Integer. maksverdien i for-løkken legges i R0)
1 MOVI R1 <- 1            (tallet som i økes med for hver runde i løkken)
2 MOVI R2 <- 0            (variabelen i lagres i R2)
3 MOVI R3 <- 0            (S = 0)
4 ADD R2 <- R2 + R1       (i++)
5 ADD R3 <- R3 + R2       (S = S + i)
6 CMP R2 R0               (COMPARE, er i = 3 ? )
7 JNE 4                   (Jump Not Equal 4, hopp til linje 4 hvis i != 3)
```

maskinkode

```
0 MOVI R0 <- 3          (MOV Integer. maksverdien i for-løkken legges i R0)
1 MOVI R1 <- 1          (tallet som i økes med for hver runde i løkken)
2 MOVI R2 <- 0          (variabelen i lagres i R2)
3 MOVI R3 <- 0          (S = 0)
4 ADD R2 <- R2 + R1     (i++)
5 ADD R3 <- R3 + R2     (S = S + i)
6 CMP R2 R0             (COMPARE, er i = 3 ? )
7 JNE 4                 (Jump Not Equal 4, hopp til linje 4 hvis i != 3)
```

Assemblykode kan enkelt oversettes til maskinkode:

Linje Nr	I Nr	DR	SR
0	0010	00	11
1	0010	01	01
2	0010	10	00
3	0010	11	00
4	0100	10	01
5	0100	11	10
6	1100	10	00
7	1111	01	00

Branch control

- Siste instruksjon i maskinkoden hopper til linje 4, avhengig av sammenligningen av to registere i forrige instruksjon.
- Gjør det mulig å utføre alle slags løkker, som for og while-løkker og i tillegg if-tester.
- JNE-instruksjonen (Jump Not Equal) gjør at man hopper til en adresse hvis sammenligningen av to registere i forrige instruksjon viste at de er forskjellige.
- Muliggjøres av branch control delen av CPU. Den gjør at PC (Program Counter) kan hoppe og ikke bare alltid øke med en.