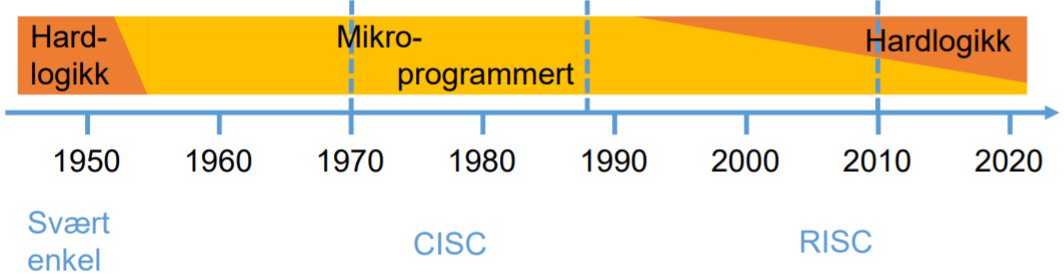
# RISC og CISC

R står for «reduced», altså mindre avanserte instruksjoner

C står for «complex», altså mer avanserte instruksjoner.

**Historikk hardlogikk V.S. mikro-programmert**

Man har gått fra å bruke hardlogikk, til mikroprogrammert grunnet begrensninger i antall transistorer. Derimot etter hvert som «Moore’s Law» har tatt effekt, bruker man mer og mer hardlogikk siden man kan ha mye mer kompliserte kontrollere med flere transistorer på samme plass som før. CISC kom på 90-tallet.



# CISC

*Den første tanken bak CISC var at med avanserte instruksjoner skulle man forsøke å tette «det semantiske gap» mellom høynivå og lavnivå programmering. Fleksible løsninger for funksjonskall som ikke begrenser programmererens arbeid.*

**Avanserte instruksjoner**

Fordel**:** Effektiv programmering (programmering er dyrt, maskinvare er billig)

Ulempe: Varierende utføringstid, vanskelig å lage effektiv pipeline siden man har en blanding av enkle og avanserte instruksjoner (få V.S. mange mikrooperasjoner). Kompliserte instruksjoner gjør også at enkle instruksjoner går langsomt. Vanskelig å bruke hardlogikk.

**Svært fleksible løsninger for funksjonskall**

* Ingen begrensinger på antall parametere, eller på antall lokale variabler.

Fordel: Legger få begrensninger på programmeringen (effektivt).

Ulempe: Hver funksjonskall krever minnetrafikk fordi parametere og lokale variabler ligger i minnet

**Mikroprogrammert kontrollenhet gir fleksibelt instruksjonsformat**

Fordel: Mye brukte instruksjoner kan være korte – noe som gir lite minnetrafikk. En programmerer kan enkelt definere instruksformat.

Ulempe: Komplisert og tidkrevende å tolke instruksjonen. Siden instruksjonene kan ha varierende lengde/format betyr at instruksjonene ikke kan tolkes parallell siden man ikke vet instruksjonsformatet da man henter dataen. (Kalles vertikal koding)

# RISC

*Denne nye trenden innen prosessorarkitektur kom på slutten av 80-tallet. De begynte designet helt på nytt, med tanke på hardlogikk grunnet økningen i transistorer*.

1. **Bruke bare enkle instruksjoner**
   * Skal være så enkel at den skal kunne implementeres i hardlogikk.
   * Hardlogikk er mye hurtigere enn mikroprogrammert kontrollenheter.
   * Antall transistorer begrenser hvor avanserte instruksjoner man kan implementere i hardlogikk.
2. **Optimaliser funksjonskall**
   * Slik at de ikke krever minnetrafikk
3. **Bruke bare enkle instruksjonsformater**

Det utvikles fremdeles CISC siden man har krav på bakover kompatibilitet.

**RISC V.S. CISC**

**ADD**

*ADD tall1, tall2 ; tall1=tall1+tall2*

CISC: tall1 og tall2 kan enten ligge i minne eller i registeret (fleksibelt). Gjør at man utføringen er mer kompleks og mer uforutsigbar.

RISC: Så enkelt og fort som mulig. Tall1 og tall2 må ligge i hvert sitt register. Hvis de ligger en annen plass (minnet), er det egne instruksjoner for dette. I tillegg hvis man vil skrive til minnet er det en egen instruksjon. Alt dette kan kanskje gjøres av en CISC prosessor med en instruksjon, men med RISC så går det oftest fortere.

**FUNKSJONSKALL**

CISC: Funksjonskall skal være fleksibel og det skal ikke legges begrensninger for programmeringen. Derfor legges parametere og lokale variabler legges i minnet (stack) før funksjonskallet.

RISC: Skal gå hurtig, vi har mange registre, returadresse, parametere og lokale variabler legges i registre.

Det viser seg at man ikke trenger enormt mange parametere og lokale variabler.

* + 98% av alle funksjoner har færre enn 6 parametere
  + 92% av alle funksjoner har færre enn 6 lokale variable.
  + I praksis er dybden av funksjonskall begrenset, eller varierer lite under programutførelsen

**INSTRUKSJONSFORMATET**

CISC: Instruksjoner varierer i lengde. Eks. x86 korteste instruksjon 1 byte, lengste 16 byte. De mest brukte instruksjonene minimaliserer minnetrafikken siden de kan være korte. Det tar lang tid å finne ut hvor instruksjonen starter og slutter, i den tiden kunne en RISC prosessor allerede ha utført instruksjonen.

RISC: Instruksjonene er like lange, gjerne 32 eller 64 bits. De har et ordinært og standardisert format. Hver felt i instruksjonen kan (som regel) dekodes uavhengig av de andre delene. Instruksjonene tar like lang tid, noe som gir effektiv pipelines.