

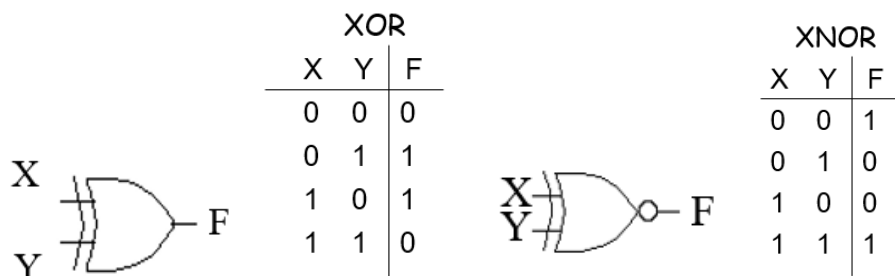
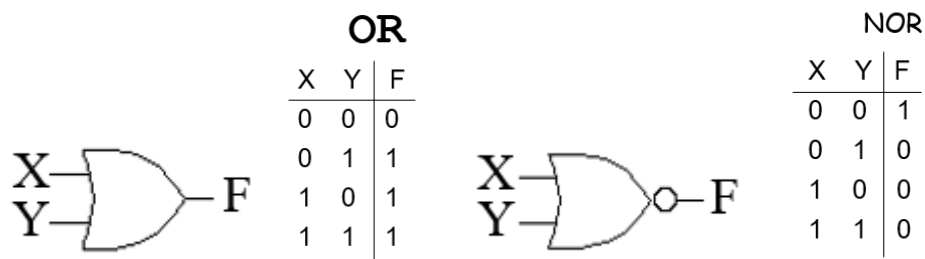
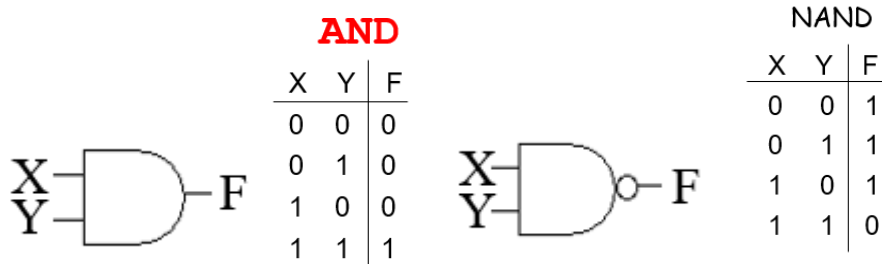
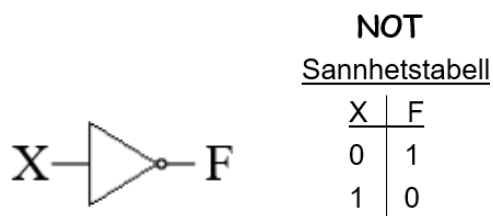
TK1100

FORELESNING 0x03 ARKITEKTUR

1) Boolsk Algebra

a) Tegn opp grunnoperasjonene(porter/gates) med tilhørende sannhetstabell

- NOT
- AND
- NAND
- OR
- NOR
- XOR
- XNOR



b) Regneoppgaver, bruk sannhetstabellene til grunnoperasjonene. Bruk penn og papir.

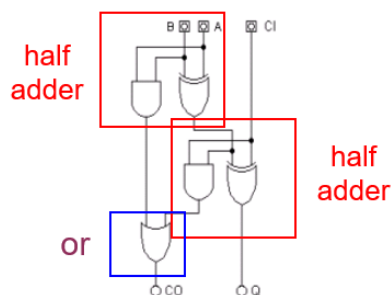
- $1100\ 0010$ **AND** $1100\ 0000 = 1100\ 0000$
- $0000\ 1100$ **OR** $0000\ 1111 = 0000\ 1111$
- $0101\ 0010$ **XOR** $1110\ 1101 = 1011\ 1111$
- **NOT** $0110\ 1111 = 1001\ 0000$
- $0000\ 1000$ **NOR** $1111\ 0000 = 0000\ 0111$
- $1111\ 1111$ **AND** $1111\ 1111 = 1111\ 1111$
- $1000\ 1001$ **NAND** $0001\ 1001 = 1111\ 0110$
- $1010\ 1010$ **AND** $0101\ 0101 = 0000\ 0000$

c) Hva brukes en 8-bits full adder til? Forklar med egne ord hvordan den fungerer.

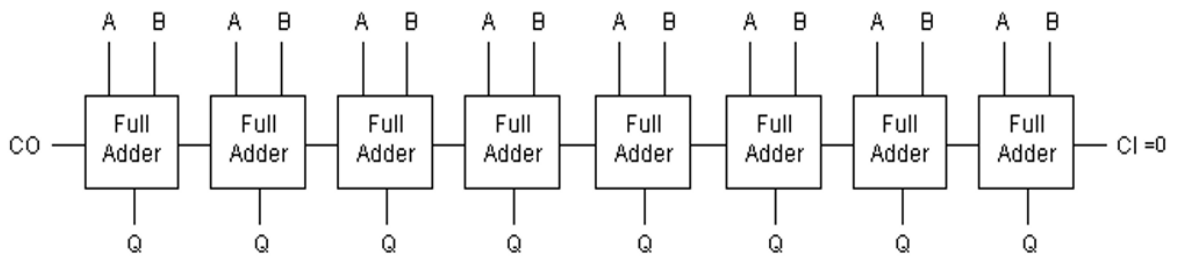
En 8-bits full adder brukes for å legge sammen tall. Når det er 8 bit betyr det at det er åtte fulle addere koplet sammen som har mulighet å legge sammen 8 bits tall. For å forklare å skjønne hvordan en 8-bits full adder fungerer så er det viktig å skjønne hvordan en full adder som kun legger sammen en bit fungerer.

Tenk litt som at du skulle regne selv, her skal vi legge sammen A og B (Se bilde under) og eventuelt de som kommer fra CI. CI er her «mente» som da kan være noe som er igjen fra forrige utregning.

A og B blir kjørt gjennom både en AND gate og en XOR gate, dette er noe som kalles for en «half adder». Svaret fra XOR kjøres gjennom en ny half adder sammen med CI (mente), og svaret fra de to AND-ene fra begge half adderene kjøres gjennom en OR port og dette blir den nye menten til neste adder, se CO i bilde under. Q er det sluttelige svaret fra en full adder.



I bilde under så vises det en 8-bit full adder, her ser vi at den har åtte «Full addere» og at CI (mente) sendes til neste adder i køen. Disse brukes altså i maskiner for å ha mulighet att legge sammen binære tall.



- d) ASCII-koden for "F" er 0x46. Hvilket tegn/glyf får vi fra ASCII tabellen dersom vi utfører: 0x46 **OR** 0x20? Bruk penn og papir for utregning.

Tegnet «f» som er 0x66 i ASCII tabellen, **OR** operasjonen her endrer kun 1 bit:

0x46 = 0100 0110

0x66 = 0110 0110

0x20 kan faktisk 'OR-es' med alle store bokstaver i ASCII tabellen, for å få små bokstaver. Prøv det gjerne ut med andre bokstaver for å se resultatet!

- e) Parr riktig logisk operator med bitwise operator.



2) Regneoppgaver

Alle oppgavene skal gjøres med penn og papir, vis utregning.

- a) Følg stegene under:

1. Finn den heksadesimale verdien til tegnet ^ i ASCII-tabellen.
2. Gjør om verdien til det binære tallsystemet.
3. Bruk AND-operasjonen med verdien du har funnet og 0110 0001.
4. Gjør om det binære tallet du står igjen med til heksadesimal.
5. Finn tegnet til den heksadesimale verdien, dette er svaret på oppgaven.

Svaret er: @

b) $(0x37 \text{ XOR } 0xF3) \text{ XOR } 0x37 = ?$ (Oppgi svar i heksadesimal)

```
(0011 0111 XOR 1111 0011) XOR 0011 0111 =  
1100 0100 XOR 0011 0111 =  
1111 0011
```

Konverter 1111 0011 til heksadesimal = **0xF3**

3) Arkitektur

a) Nevn noen av de viktigste elementene på ett hovedkort.

Svaret burde være innom disse punktene:

- CPU - Prosessor
- North bridge – kobler sammen prosessor, minne og GPU
- South bridge – direkte koblet til North bridge og resterende komponenter på hovedkortet, eks. I/O enheter og BIOS
- BIOS – Inneholder alt for oppstart av maskinen
- Ulike slots for å koble RAM, periferiutstyr, harddisk etc., eksempel PCI, SATA, IDE.

b) Gi eksempler på hva som kan kobles til et hovedkort.

Her finnes det mange riktige svar, eksempel:

- GPU (Graphics Processing Unit), grafikkort
- Harddisk
- Minne, eks. RAM
- Lydkort
- Nettverkskort
- Mus
- Tastatur
- Printer
- Ekstern skjerm
- Ekstern harddisk
- Etc.

c) Forklar hva en GPU er.

En GPU er enten direkte tilkoblet til hovedkortet eller på et grafikkort som senere blir koblet til hovedkortet, det er GPU-en som prosesserer signaler fra maskinen til skjermen. Dette er altså en mikroprosessor som er designet for å utføre grafikkrelaterende beregninger, dette er altså ikke noe som CPU-en utfører.

4) CPU

a) Hva er en instruksjon?

Det er instruksjonen til CPU-en i en maskin som forteller hvilke oppgaver den skal og hva slags data den skal utføre det med. Man sier altså til maskinen hva den skal gjøre med hjelp av instruksjoner.

Hvilke typer av instruksjoner en prosessor har mulighet til å utføre er klarlagt i et så kallet instruksjonssett, eksempel x86.

b) Forklar forskjellene mellom en CISC-prosessor og en RISC-prosessor.

CISC betyr Complex Instruction Set Computer og RISC betyr Reduced Instruction Set Computer. Gjennom navnene kan man skjønne at CISC er for mer komplekse system (desktop og server) og RISC er for mindre system som er enklere og trenger en veldig rask utførelse fra prosessoren (embedded og mobil). CISC og RISC er to forskjellige arkitekturer av en prosessor, her finnes det i dag også CPU-er som bruker seg av begge arkitekturene, eksempel moderne Intel/AMD CPU-er hvor CISC er eksternt og RISC er på selve prosessoren i maskinen. Andre produsenter av blant annet prosessorer er MIPS og ARM, de fokuserer på RISC arkitektur.

Maskiner som bruker RISC utfører en instruksjon per klokkesyklus og CISC maskiner kan ha instruksjoner som bruker mer enn en syklus for å bli utført. Når man bruker RISC trengs det mer RAM minne enn når man bruker CISC. Det vil si at CISC bruker RAM mer effektivt. En instruksjon på en CISC maskin kan trenge flere instruksjoner på en maskin med RISC arkitektur da denne kun utfører en instruksjon per klokkesyklus.

c) Finn ut av hva slags type prosessor du har i din PC og telefon, og hva slags ISA den prosessoren har.

Her er noen guider på hvordan finne prosessor i PC-en:

Windows:

<https://www.howtogeek.com/413942/how-to-see-what-cpu-is-in-your-pc-and-how-fast-it-is/>

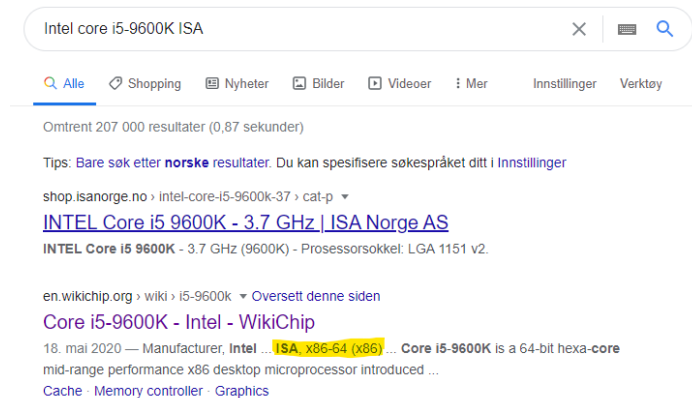
MacOS:

<https://www.techjunkie.com/find-mac-cpu-model/>

Linux:

<https://www.cyberciti.biz/faq/check-how-many-cpus-are-there-in-linux-system/>

Etter du har funnet ut navnet på cpuen skal det være lett å google seg frem til hvilken ISA den har. Typ:



d) Hva er en mikrokontroller og hvor brukes det?

Typisk eksempel på hvor man bruker mikrokontroller er embedded systems, her er det mikrokontrolleren som styrer systemet hvor den ofte kun kjører et program, som gjerne kan være enkelt. Det finnes mange ulike embedded system, de vi kjenner godt til er vaskemaskiner, klokkeradioer, hjemme-routere etc. En mikrokontroller er altså ett alternativ til CPU. Her er den store forskjellen at en CPU er kun en prosessor og bruker seg av operativsystem, men mikrokontroller har også minne, I/O, lagring og bruker sjeldent operativsystem.

e) Skriv kort om x86 instruksjonssett.

Svaret burde være innom disse punktene:

- Eksempel på hvor det brukes, for eksempel prosessorer med CISC arkitektur
- Hvorfor det finnes instruksjonssett, for at forstå at en prosessor trenger retningslinjer på hvordan den skal agere.
- Hvilke deler x86 inneholder (Data flytting, Kontrolloverføring, Aritmetikk/Logisk, Input/Output, Debug og Interrupt håndtering)

5) Teknikker og begreper

a) Hva er forskjellen på en system-klokke og en vanlig klokke.

En system-klokke viser ikke tiden, den holder takten i et system for at komponentene på hovedkortet skal være synkronisert og sikkert, hvilket betyr at de utfører sin oppgave når system-klokken er på «high» og ikke når den er på «low».

b) Når du kjører et program på en datamaskin, hvor ligger det fulle programmet?

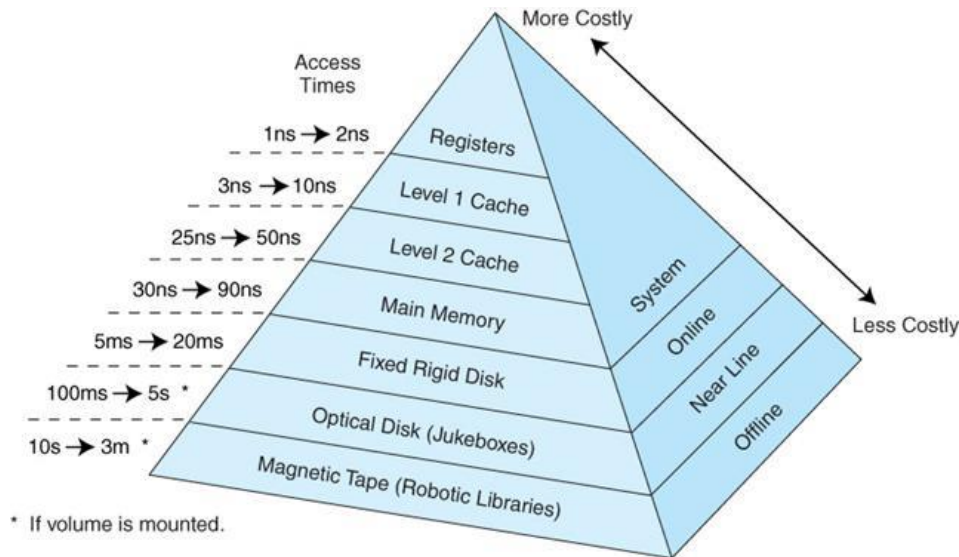
Når du kjører et program så vil det fullstendige programmet ligge i RAM-en, her er det kun enkeltinstruksjoner og enkeltdata som ligger på CPU-en når programmet er i gang. CPU-en får altså instruksjoner og data fra RAM-en og disse går via North bridge for å komme frem til CPU-en.

- c) Hvorfor har en CPU flere kjerner?

For å gjøre det mulig å kjøre flere program samtidig og at det skal gå raskt. Se for eksempel hvor mange program du har oppe og kjører akkurat nå, dette hadde ikke CPU-en klart hvis den ikke hadde hatt flere kjerner.

- d) Hva representerer minnehierarkiet?

I minnehierarkiet er det mulighet til å få forståelse og oversikt av hvilken type av minnelagring som er enkel å aksessere, mest sparsom og mest effektiv.



- e) Ut ifra minnehierarkiet hva slags type lagring ville du brukt om du skulle lagret noe du bare trengte tilgang til hvert 5. år.

Her kan svar på oppgaven variere, det som er viktig er å forstå forskjellene mellom de ulike typer av lagring og hva som er det positive/negative med de. På grunn av at denne informasjonen kun skal hentes ut hvert 5. år så er en type offline løsning et bra alternativ, dette kan for eksempel være en ekstern harddisk.

- f) Hva brukes cache-minnet til?

Det tar tid før CPU-en å få informasjon fra RAM-en, så for at det skal gå kjappere å få frem informasjonen så finns cache-minne som ofte er lokalisert direkte på CPU-en. Når CPU-en kjører et program å skal hente informasjon så sjekker den først cachene for å se om de den søker finnes der, om ikke så sjekker den i RAM-en. I dag er det vanlig med flere cache-minner, disse kan ha ulike spesifikasjoner og effektivitet.

- g) Hva går en CPU gjennom for å hente data fra RAM-en.

For å hente informasjon fram RAM-en trenger CPU-en å gå gjennom North bridge, her er det North bridge som er direkte koblet til både CPU-en og RAM-en.

6) Repetisjonsspørsmål

- a) Vi arbeider med 8 bits presisjon og toerkomplement. Utfør binærsubtraksjonen:
 $1000\ 0111 - 0010\ 0001 = ?$

Svar: 0110 0110

- b) Foreta en logisk OR mellom 1001 1111 og 0101 1011

Svar: 1001 1111 OR 0101 1011 = **1101 1111**

- c) $1001\ 1111 \& 0101\ 1011 = ?$

Svar: 0001 1011

- d) Hva blir resultatet dersom man subtraherer («trekker») ASCII-koden for tegnet «3» fra ASCII-koden for tegnet «0»? Oppgi svaret i heksadesimal.

Svar: 0xFD

- e) Hva er UTF-8 kodingen av Unicode-tegnet U+00BD (1/2)?

- 0xBD
- **0xC2BD**
- 0x00BD
- 0xBD00
- Ingen av alternativene over

- f) Hva blir resultatet av den binære addisjonen $0100\ 1001 + 0101\ 0101$?

- **1001 1110**
- 1100 0000
- 1100 1111
- 0100 0001
- 0111 1111

- g) Hvordan vil det negative tallet 53 (-53) bli lagret som en byte?

- 0101 0011
- 1101 0101
- 1100 1010
- **1100 1011**
- 1101 0110