Boolske operatorers sannhetstabeller

1. Fyll ut sannhetstabellen til NOT

NOT A = Q

- NOT 0 = ___1__
- NOT 1 = ___0__

2. Fyll ut sannhetstabellen for logisk (bitwise) AND

A and B = Q

- 0 AND 0 = __0___
- 0 AND 1 = __0___
- 1 AND 0 = __0___
- 1 AND 1 = __1___

3. Fyll ut sannhetstabellen til OR

A OR B = Q

- 0 OR 0 = ___0__
- 0 OR 1 = ___1__
- 1 OR 0 = ___1__
- 1 OR 1 = ___1__

4. Fyll ut sannhetstabellen til XOR

A XOR B = Q

- 0 XOR 0 = ___0__
- 0 XOR 1 = ___1__
- 1 XOR 0 = ___1__
- 1 XOR 1 = ___0__

5. **Gitt at sannhetstabellen for AND er:**

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Hva er 1010 0100 AND 0101 1011?

- 1111 1111
- 1010 0000
- 0000 0000
- 0000 0001

6. **Sannhetstabellen for logisk OR er:**

A	B	A AND B
0	0	0
0	1	1
1	0	1
1	1	1

Hva er 1010 0100 OR 0101 1011?

- 1111 1111
- 1010 0000
- 0000 0000
- 0000 0001

7. **Sannhetstabellen for NOT er:**

A	NOT A
0	1
1	0

Hva er NOT 1001 1100?

- 1111 1111
- 0000 0000
- 0110 0101
- 0110 0011

8. **Sannhetstabellen for logisk XOR er:**

A	B	A AND B
0	0	0
0	1	1
1	0	1
1	1	0

Hva er 1100 XOR 0110?

- 1000
- 0110
- 1010
- 0011

Boolske anvendelser

1. **ASCII-koden for "A" er 0x41.**

   Hvilket tegn/glyf får vi ASCII-koden for dersom vi utfører:

   0x41 OR 0x20

   - 0x61
   - a
   - æ
   - Ingen av alternativene over

2. **Vi kan bruke OR med "bitmasken" 0x20 til å ... (flere alternativer kan være korrekt)**
   - sørge for at ASCII-tegnet for en stor bokstav blir til ASCII-tegnet for en liten bokstav.
   - sørge for at ASCII-tegnet for et tall blir til det tilsvarende binærsifferet.
   - sørge for at det TREDJE sifferet **fra venstre** i en byte ("det tredje mest signifikante") blir til **1**uansett hva det var før operasjonen.
   - Ingen av alternativene er korrekte

3. **Dersom du tar AND mellom ASCII-koden 0x37 (tegnet 7) og 0xCF så får binærsifferet ...**
   - 1111 1000
   - 0000 0111
   - 1100 1111
   - Ingen av alternativene er korrekte

4. **Dersom du vil være sikker på at alle bokstavene i en setning er store så kan du AND'e hvert ASCII-tegn i setningen med 1101 1111**
   - Enig
   - Uenig

5. **For å bytte frem og tilbake mellom liten og stor bokstav (ASCII-tegn) trenger vi bare å gjenta operasjonen ___0x20__ hver gang vi vil bytte om.**
   - 0xFF
   - 0x20
   - 0x2F
   - 0x10

Du vil sikkert ha nytte av https://paulschou.com/tools/xlate/ og kalkulatoren i Hex-modus når du skal løse oppgaven under. Det er også greit å ha åpent et vindu med ASCII-tabellen

6. **Dersom du tar XOR mellom hvert enkelt ASCII-tegn i meldingen [vz2 og kodenøkkelen 0x13, så får vi frem meldingen** __Hei!___

   (Du må altså

   **** finne hvilket hexadesimalt tall som svarerer til tegnene [, v, z, 2;***

   **** så må du utføre XOR med 0x13 på hver av disse, da får du fire nye hexadesimale tall***

   **** som du må slå opp i ASCII-tabellen for å finne hvilket tegn det hexadesimale tallet koder.)***

   **** XOR er faktisk veldig mye brukt til koder...***

7. **(0x37 XOR 0xF3) XOR 0x37 =** __0xF3___ (kan være både 0xF3 og 1111 0011)

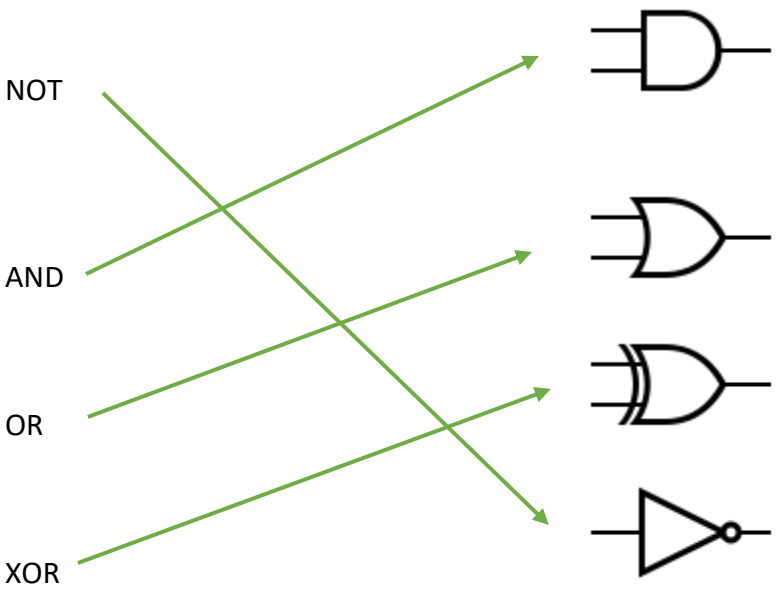
8. **NOT gjør nøyaktig det samme som XOR med 0xFF, gitt åtte bits presisjon.**

   M.a.o. NOT(0xFF) == (0xFF XOR 0xFF);

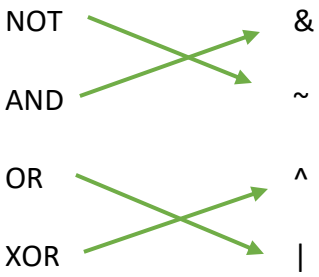
   (eller i pseudo-Java (~0xFF) == (0xFF^0xFF) vil evalere til sann.)

   - True
   - False

9. Sett sammen de logiske operatorene (navnene) til venstre med korrekt symbol (til høyre)



10. Parr riktig logisk operator med hvilken bitwise operator som brukes for denne i Java.



Begreper og teori

1. **Hva lagres i arbeidsminnet (RAM)? (Kryss av for alle riktige svar!)**
   - (Minne-)adresser
   - Data
   - Kontrollsignaler
   - Instruksjoner
   - Dekodingsregistre
   - Program
   - Murstein

2. **Register er betegnelsen på**
   - lagringsplasser på en byte som befinner seg på en harddisk. De er typisk laget på samme måte som DRAM, men er magnetiske i stedet for elektriske.
   - lagringsplasser (minneceller) som er tilgjenglige på CPUen. De er typisk laget på samme måte som SRAM.

3. **Hvilken av disse enhetene utfører aritmetiske og logiske operasjoner på heltall?**
   - FPU
   - MMU
   - HAU
   - ALU
   - CLUedo
   - Ingen av alternativene ovenfor

**Sett inn alternativene under på rett plass*

4. **___x86__ er betegnelsen på en familie instruksjonsett ("ISA") for CPUer som startet med Intel sin ___8086__ prosessor i __1978___**
   - x86
   - 8086
   - 1978

**Sett inn alternativene under på rett plass*

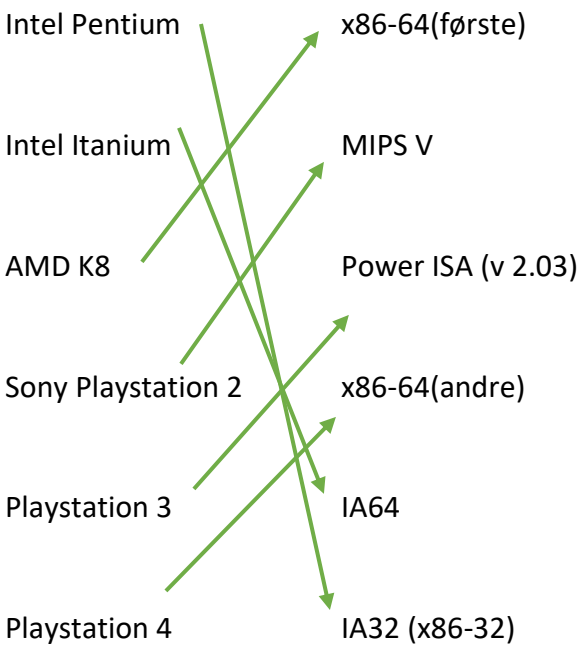
5. __Apple Macintosh___**benyttet opprinnelig** ___Motorola 68k__ **prosessorer, gikk så over til** ___PowerPc__ **, før de til slutt gikk over til** ___x86__ **typen i** __2005___
    - Apple Macintosh
    - Motorola 68k
    - PowerPC
    - X86
    - 2005

**Sett inn alternativene under på rett plass*

- **En GPU er en spesiell type** __Mikroprosessor___ **som utfører SIMD** __(Single Instruction Multiple Data)___ **på vektorer og** __matriser___ **. De brukes dermed til** ___grafikk__ **-behandling.**

- Mikroprosessor
- (Single Instruction Multiple Data)
- Matriser
- grafikk

6. **Parr riktig CPU/system med riktig instruksjon-sett type**



7. **Marker de trinnene som (vanligvis) kan inngå i en Fetch-Execute Syklus**
    - Innlesing av instruksjon fra minnet (RAM) til CPU
    - Innlesing av programmet til RAM fra harddisk
    - Dekoding av instruksjonen og oppsett av logikken som skal utføre den
    - Utførelse av instruksjonen
    - Tilbakeskriving til minne av resultatet

8. **Systemklokken på CPUen**
    - bestemmer når hvert trinn i Fetch-excute syklusen gjennomføres
    - holder rede på årstall, måned, dag og tidspunkt.

9. **Pipelining ...**
    - en teknikk for å kunne utføre fetch-execute-syklusen i parallell for flere instruksjoner "samtidig"
    - er det samme som supereskalaritet
    - forutsetter at man har flere prosessorkjerner på CPUen
    - gjør at CPUen arbeider sakterte

10. **Minne-hierarkiet**

    Stabl elementene under slik at dedyreste, raskeste og minste ligger til venstre, de største og treigeste (med lengst aksesstid) ligger til høyre

Register	L1 Cache	L2 Cache	RAM	SSD	Harddisk	DVD

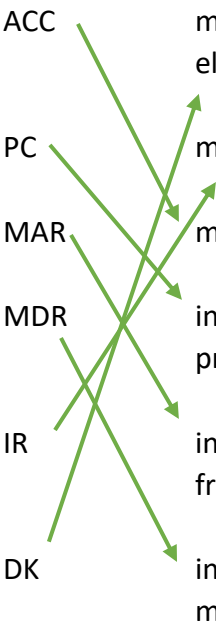
CPU simulator

Etter å ha startet opp programmet skal du få opp skjermbildet under:

1. Velg "Demonstrasjon".
2. Du får da opp et skjermbilde som viser deg hvordan dette programmet ville ha sett ut i 16-bit Intel-assembly.
   Velg "Forsett til programmet".
3. Du skal nå ha fått opp en modell av den enkle prosessoren som CPU-applikasjonen simulerer.
4. Trykk på start-knappen nederst til venstre:

5. Følg med på det som skjer og les "Hendelsesloggen"

1. **Hvilke registere har hvilken rolle i kjøringen av programmet på CPUen? (Koble sammen riktige utsagn)**



ACC	mellomlagrer opkoden slik at den kan brukes til å aktivere den elektroniske kretsen som utfører instruksjonen.
PC	mellomlagrer instruksjonen som skal dekodes
MAR	mellomlagrer variabelverdier (data) og resultater av regneoperasjoner.
MDR	inneholder minneadressen til neste instruksjon som skal hentes inn fra primærminnet (RAM).
IR	inneholder minneadressen til data eller instruksjoner som skal hentes inn fra minnet.
DK	inneholder data (variabelverdier) som er lest fra, eller skal skrives til minnet.

2. **Klikk på instruksjonen i primærminnet som lagrer resultatet av addisjonsoperasjonen**

3. **Klikk på minnecellen som inneholder adressen som resulatet av addisjonen i "Demonstrasjonen" lagres på.**

4. **Hvilken (hexadesimal) minneadresse lagres resultatet av addisjonen på?**
   - 0x6
   - 0xF
   - 1111
   - Ingen av alternativene over

1. **Skriv et program som trekker 19 fra 15 og lagrer resultatet på minneplass 0x9.**

   Test at det lar seg kjøre på simulatoren og gir forventet resultat!

   (NB! Skriv inn koden du selv legger inn: LOAD, SUB, STORE, STOP.

   Ikke oversettelsen til 16 bit Intel-assembly som spretter opp når du skal prøve å kjøre programmet!

   LOAD 19

   SUB 15

   STORE 9

   STOP

2. **Benytter denne CPUen "pipelining"?**
   - Ja
   - Nei

   **Sett inn alternativene under på rett plass*

3. **Hva er hensikten med STOP-instruksjonen?**

   Instruksjonssettet må ha en ___STOP instruksjon__ fordi ellers vil __programtelleren___ blitt inkrementert (talt opp) videre...

   CPUen ville da ha hentet inn og kjørt __de binære dataene___ og adressene fra ___RAM__ (primærhukommelse) som om de var __instruksjoner___

   Det ville ført til et __kræsj___ føre eller siden.

   På en virkelig prosessor vil man avsluttet med en instruksjon som returnerte kontroilen til Operativsystemet.

   - STOP instruksjon
   - de binære dataen
   - instruksjoner
   - RAM
   - programrelleren
   - kræsj

Repetisjon

1. **Vi arbeider med 8 bits presisjon.**
   Utfør binærsubtraksjonen:
   - 1000 0111 - 0010 0001 = __0110 0110___

2. **Foreta en logisk OR mellom 1001 1111 og 0101 1011**
   - 1001 1111 | 0101 1011 = __1101 1111___

3. **Hvor mange adresselokasjoner kan adresseres dersom start- og stoppadressene er som angitt: 0x3CA8 og 0x3CFF?**
   (Svaret og mellomregningene skal oppgis hexadesimalt!!!
   Husk at å trekke fra er det samme som å legge til toerkomplementet!!!)

   (__0x3CFF___ - __0x3CA8___) + 1 = 0x3CFF + __0xC359___ = __0x0058___

4. **Hvor mange byte er det i en MiB?**
   - 1 000
   - 1 024
   - 1 048 576 (1024*1024)
   - Ingen av alternativene over

5. **Hva blir resultatet dersom man subtraherer («trekker») ASCII-koden for tegnet 3 fra ASCII-koden for tegnet 0 (0x30)?**
   - -2
   - 0xFD
   - 0x36
   - 0xCD
   - Ingen av alternativene over

6. **Hva er ASCII-koden for b (liten b)?**
    - 48
    - 0x42
    - 97
    - 98
    - Ingen av alternativene over

7. **Gitt at man bruker ordbok-koding med kode 0 for X, 1 for Y og 2 for mellomrom. Hvilken melding blir komprimert som 00123012344?**
    - XXY XY XXY XXY XXY
    - XXY XXY XY XXY XY XY
    - XY XY XYZ XYZ XYZ XY
    - XXY XY XXY XXY XYX YXY
    - Ingen av alternativene over

8. **I minne-hierarkiet vil?**
    - L2 Cache plasseres høyere opp enn harddisk.
    - SSD (Solid State Disk) plasseres høyere opp enn L2-cache.
    - L3 Cache plasseres høyere opp enn L2 fordi det er mer av det.
    - Minne-pinne plasseres lavere enn Tape fordi den har lavere kapasitet.
    - Ingen av alternativene over

9. **Hva er UTF-8 kodingen av Unicode-tegnet U+00BD (½)?**
    - 0xBD
    - 0xC2BD
    - 0x00BD
    - 0xBD00
    - Ingen av alternativene over

10. Hva er Fetch-execute-syklusen?

- En måte å benytte akkumulator-registeret på slik at data ikke trenger å skrives tilbake til minne før de er ferdig behandlet på CPU.
- Et superskalart alternativ til pipelining som benyttes i GPUer.
- De trinnene en CPU går gjennom fra den henter en instruksjon fra minnet til instruksjonen er ferdig utført.
- Mellomlagring av instruksjoner og data, typisk for å oppnå hastighetsøkning i systemet.
- Ingen av alternativene over.

11. Hva blir resultatet av den binære addisjonen 0100 1001 + 0101 0101?

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

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

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

13. Hvilker farger er grunnfargene i en skjerm?

- Sort og Hvit
- Rød, grønn, blå
- Cyan, magneta og gul
- 256
- Gul, blå og hvit

14. va er resultatet av en logisk AND mellom 0011 og 1011?

- 1001
- 1011
- 0011
- 0110
- 1100
- Ingen av alternativene

15. Hva er resultatet av en logisk eksklusiv eller (XOR) mellom 1001 xor 0101

- 1101
- 1011
- 0001
- 1100
- 1001
- Ingen av alternativene over