Agenda

Første forelæsning:

- Præsentation af A2
- Nedstigning! Fra store til små byggeklodser

Anden forelæsning:

- Nedstigning til CMOS transistorer, Fremstillingsprocess, Moores lov
- Internationaliserings-reklame-fremstød

A2 - Simulering af x86prime

- Vi giver jer en simulator der ikke er helt færdig
- En simulator er et program som lader som om det er en maskine. Det vil sige: det kan udføre andre programmer skrevet til den maskine der simuleres.
- Senere giver I os en simulator der *er* færdig. Den kan udføre x86prime programmer
- Der er krav til *hvordan* i skriver jeres simulator. De krav afspejler hvordan hardware essentielt virker.

Modellering i C

Vi har oversat egenskaberne ved byggeklodserne og hvordan de kombineres til *formkrav* til programmer skrevet i C. Det ligner ikke normalt C.

Vi bruger kun to typer i vores program: en til kontrol-signaler, en til data:

- Vi repræsenterer et kontrol signal med en "bool"
- Vi representerer øvrigt data med en "val"

En maskin cyklus udtrykkes ved et løkke gennemløb. Tilstandselementer erklæres udenfor løkken. Alle de funktionelle byggeklodser udfører deres arbejde først i hvert gennemløb. Alle tilstandselementer opdateres i slutningen af hvert gennemløb.

```
ip_reg_p ip_reg = ip_reg_create();
while (..) {
  // funktionelle byggeklodser:
  val ip_out = ip_read(ip_reg);
  val ip_next = add(ip_out, from_int(1));
  // clock-puls... opdatering af tilstandselementer:
  ip_write(ip_reg, ip_next, true);
}
```

Udleveret kode - Forbindelser

```
// simple conversion
val from_int(uint64_t);

// pick a set of bits from a value
val pick_bits(int lsb, int sz, val);

// pick a single bit from a value
bool pick_one(int position, val);

// sign extend by copying a sign bit to all higher positions
val sign_extend(int sign_position, val value);
```

Og flere, men ovenstående er de mest betydende

Udleveret kode - Logiske operationer

For kontrol signaler, repræsenteret ved bool's bruger man bare de indbyggede logiske operatorer.

For data, repræsenteret ved typen val finds der følgende funktioner

```
// mask out a value if control is false
val use_if(bool control, val value);

// bitwise and, or, xor and negate for bitvectors
val and(val a, val b);
val or(val a, val b);
val xor(val a, val b);
val neg(int num_bits, val);

// reduce a bit vector to a bool by and'ing or or'ing all elements
bool reduce_and(int num_bits, val);
bool reduce_or(val);

// 64 bit addition
val add(val a, val b);
```

Vi udleverer også hele beregnings-enheden (compute-unit).

Fejlfinding

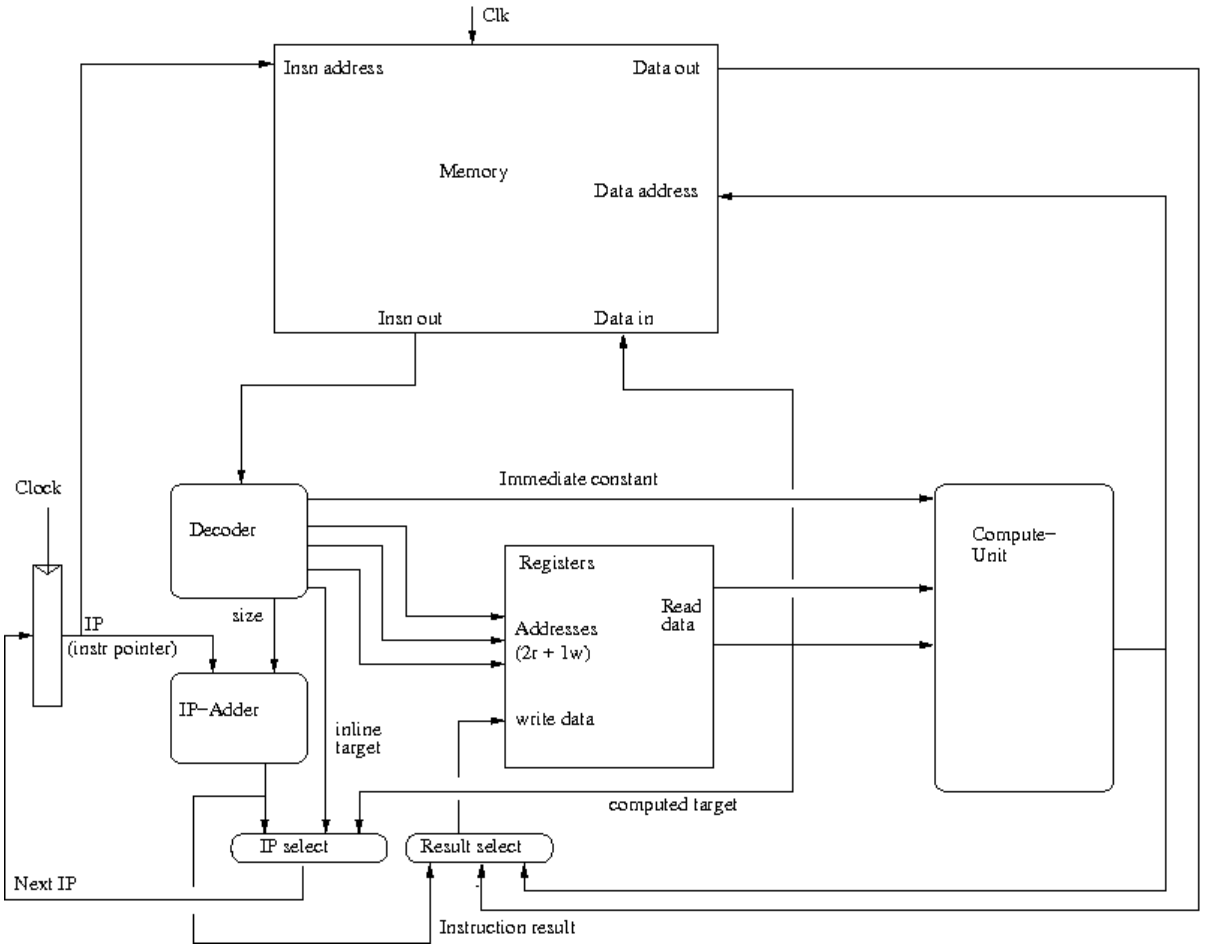
x86prime (også kaldet reference-simulatoren) kan producere en sporings-fil - en fil med de sideeffekter et program har mens det udføres.

Den udleverede simulator til a2 (den I skal færddiggøre) kan læse sådan en sporings-fil og sammenholde den med hvad der sker under simulationen.

Hvis der detekteres en afvigelse fra "sporet", kaldes en funktion der hedder error() med en fejlmeddelelse. Meddelelsen skrives ud og programmet terminerer.

Men man kan køre simulatoren i gdb og sætte et breakpoint på error(). Derefter er det ligetil at inspicere variable i programmet

A2 mikroarkitektur



Oversigt over forelæsningerne i maskinarkitektur

1. En essentiel maskine bygget af nogle passende byggeklodser. Så simpelt som muligt - men ikke simplere. A2.

2. Et deep dive i hvordan klodserne bygges. Masser af detaljer der giver baggrund

3. Pipelining - hvorfor og hvordan? Performance! Mere performance! Hvor langt kan man gå? Hvordan? Mere realisme

4. Avanceret mikroarkitektur. Parallel udførsel af sekventiel kode. Hvordan? "The bureaucracy is expanding to meet the need of the expanding bureaucracy"

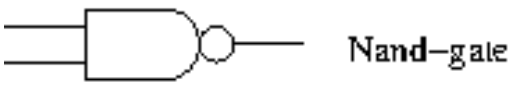
Abstraktionsniveuaer

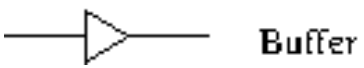
1. Højniveau programmeringssprog: Erlang, OCaml osv
2. Maskinnære programmeringssprog: C
3. Assembler / Symbolsk Maskinsprog: x86, ARM, MIPS
4. Arkitektur (ISA): Maskinsprog - ordrer indkodet som tal
5. Mikroarkitektur: ting som lager, registre, regneenheder, afkodere og hvordan de forbindes så det bliver en maskine
6. Standard celler: Simple funktioner af få bit (1-4) med et eller to resultater. Lagring af data (flip-flops)
7. Transistorer
8. Fysik. Eller noget der ligner

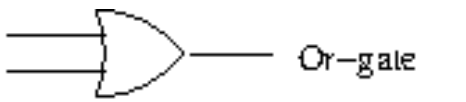
Porte. De simpleste byggeklodser

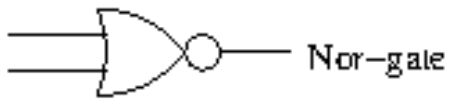
A	B	(A and B)	(A nand B)	(A or B)	(A nor B)	(A xor B)
0	0	0	1	0	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	0	1	0	0

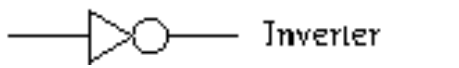
A	not A
0	1
1	0



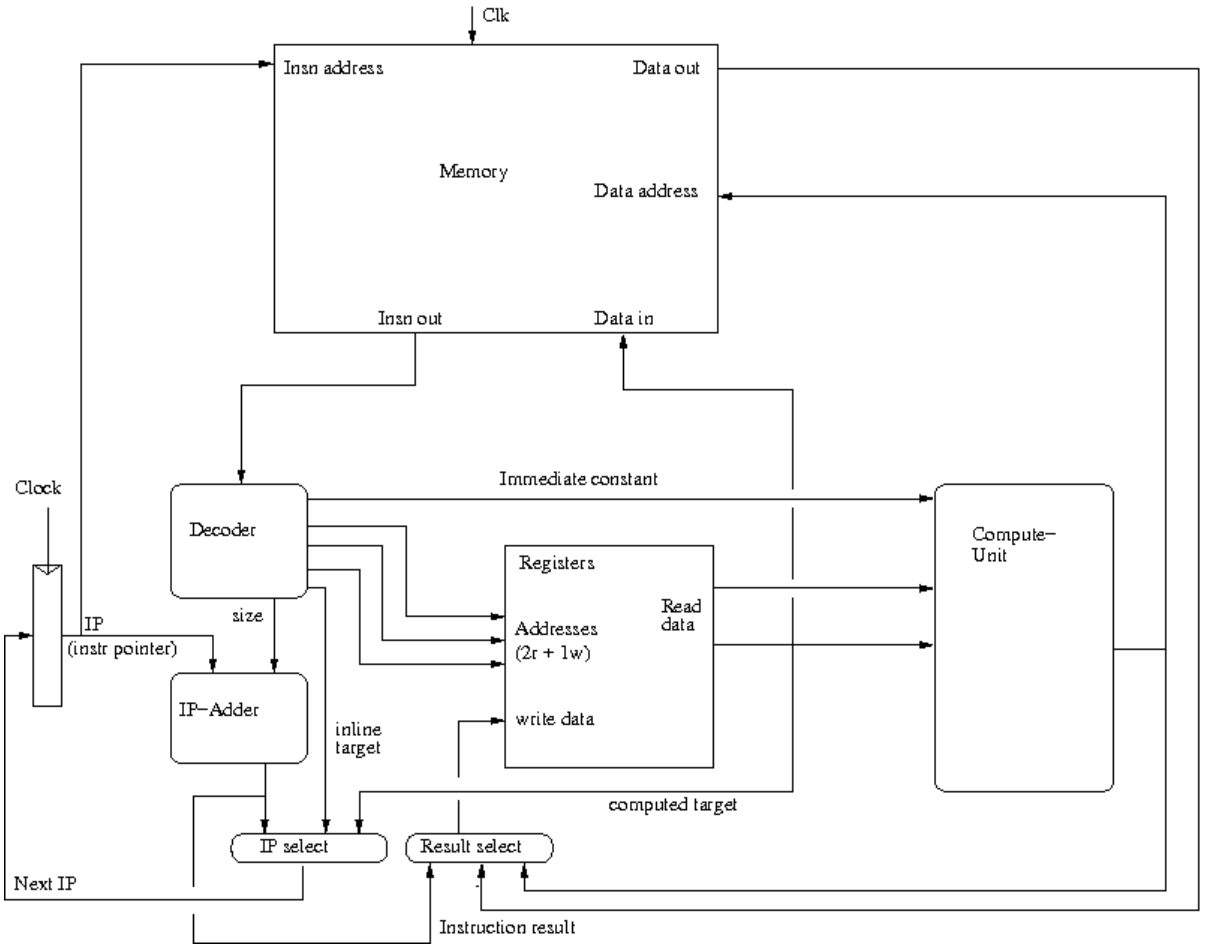




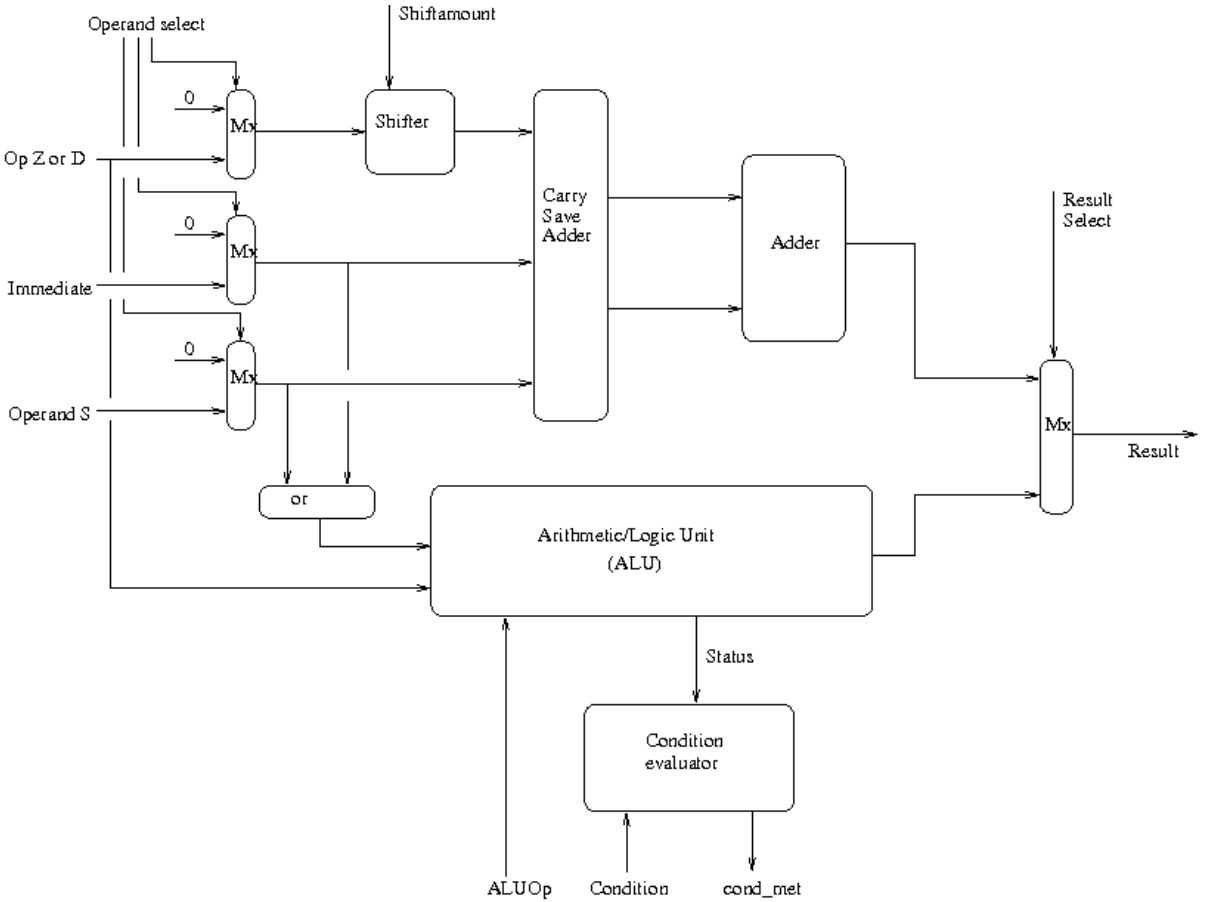




Store klodser laves af små klodser

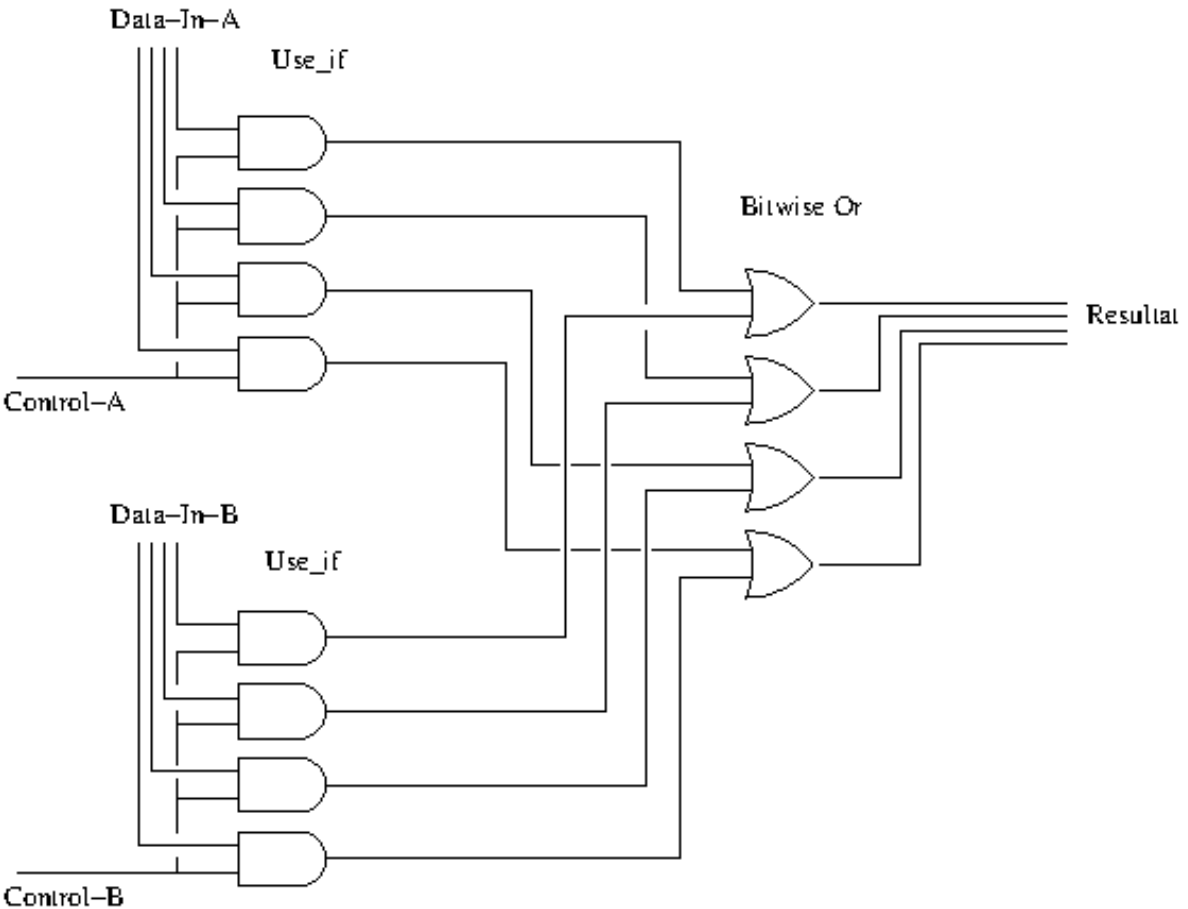


Store klodser laves af små klodser (II)



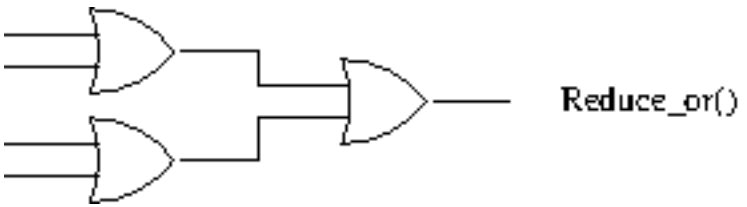
Sådan laves en multiplexor

Ved hjælp af use_if() og or()



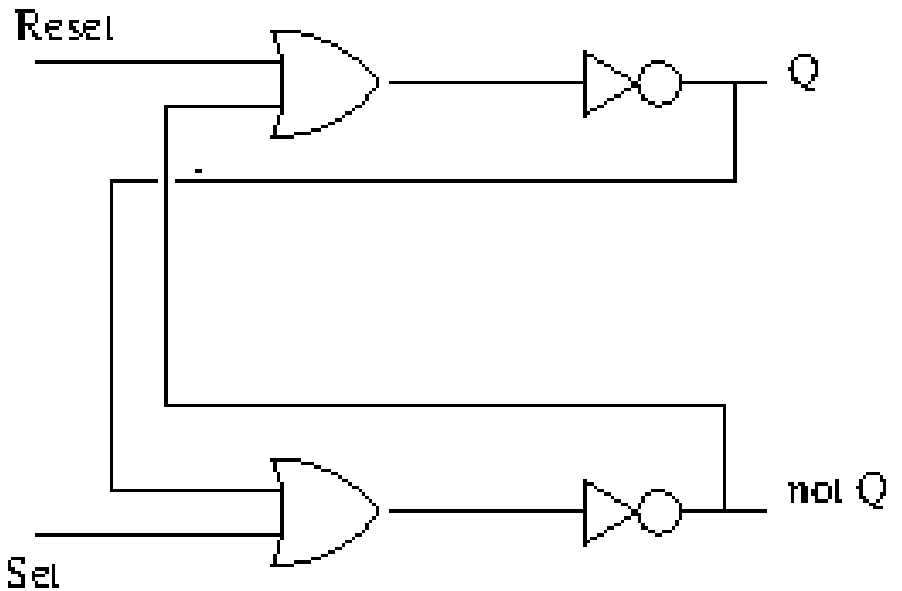
Sådan laves en "er lig med"

```
val is_equal(val a, val b) {
    val bitwise_diff = xor(a, b);
    bool equal = ! reduce_or(bitwise_diff);
    return equal
}
```



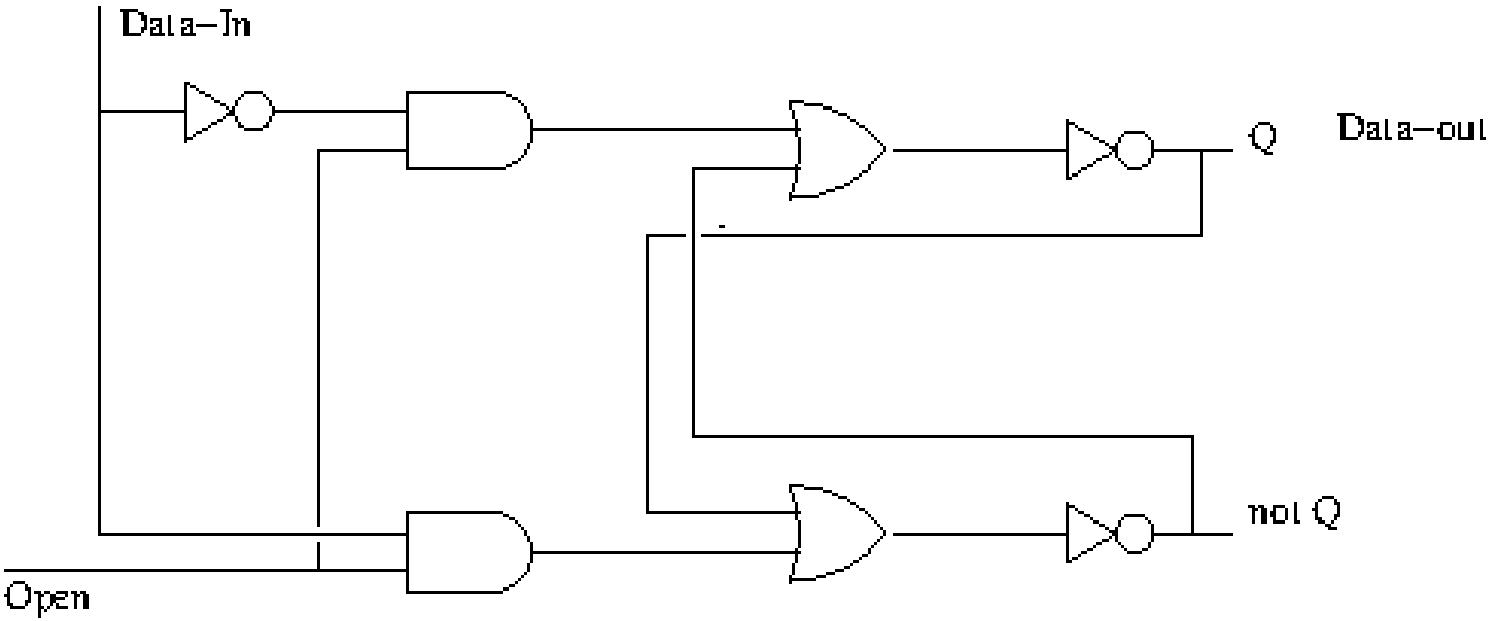
Lagring af en enkelt bit

En RS latch (R=reset, S=Set)



Lagring af en enkelt bit (II)

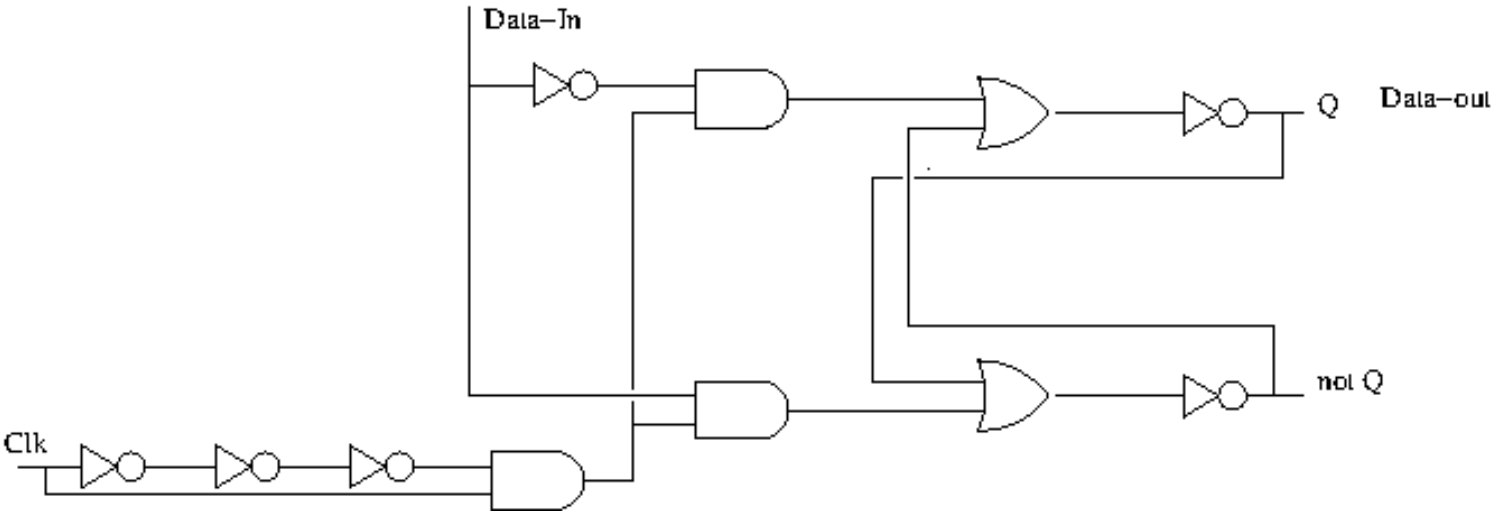
En D-latch (D for Data). Opdateres sålænge 'open' er høj



Undertiden kaldes en D-latch også for en SRAM-celle.

Lagring af en enkelt bit (III)

En Kant-trigget D-latch. Opdateres på den stigende flanke af 'clk'



Et helt register kan bygges af D-latche. En til hver bit.

En lager blok - Skriveport

En lagerblok består af et antal registre.

Clk signalet til hver register betinges af at netop dette registers nummer er magen til det register nummer der drives på addresse indgangen.

Data på data indgangen føres frem til samtlige registre.

En lager blok - Læseport

En stor multiplexor udvælger den ønskede værdi. Pseudokode for en lager blok med 4 registre:

```
val read_reg(val addr) {
    return or( or( use_if( is(0, addr), reg_0),
                   use_if( is(1, addr), reg_1))
               or( use_if( is(2, addr), reg_2),
                   use_if( is(3, addr), reg_3)));
}
```

Det skalerer ikke særlig godt. Så for store lagerblokke skal der andre metoder i brug.

Implementation af porte

Den grundlæggende funktionalitet kan implementeres på et utal af måder

Man kan f.eks. bruge tryk (luft eller vand) til at repræsentere 1 og mangel på tryk til at repræsentere 0. Eller omvendt. Og så bygge en hel computer op omkring det. Blot man kan lave en ventil der kan styres af tryk, så er den ged barberet.

Man kan også finde nogle klodser i Minecraft og sætte dem sammen.

Men vi vil interessere os for implementationer i CMOS: Complementary Metal Oxide Semiconductors.

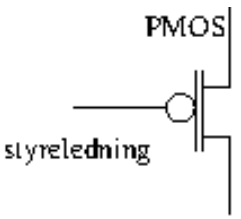
Transistorer

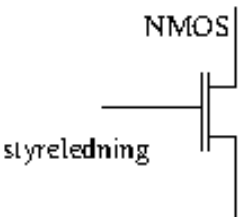
En transistor i CMOS er en kontakt som kan styres af en spænding.

Der er to slags transistorer.

- PMOS: kontakten er åben, når der ikke er spænding på styreledningen
- NMOS: kontakten er åben, når der er spænding på styreledningen

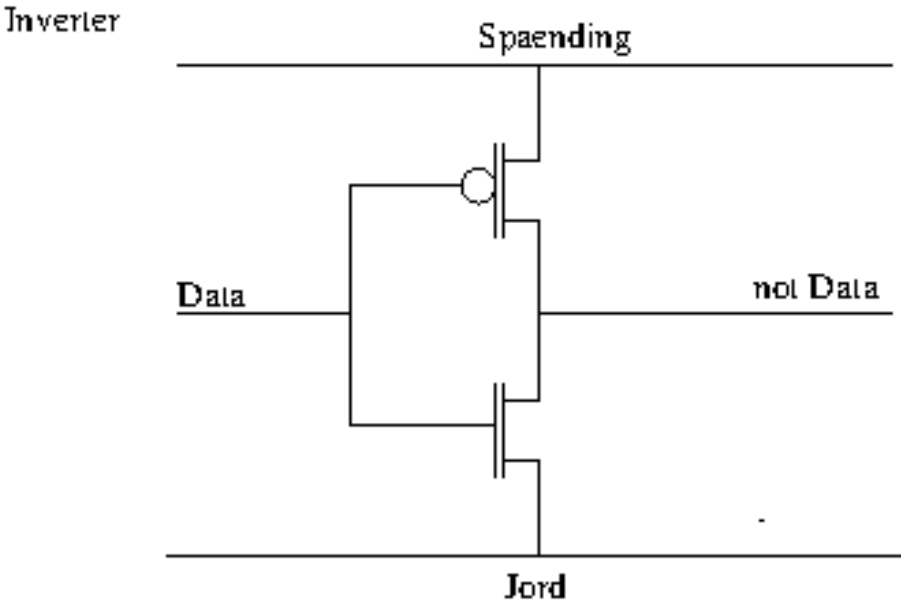
I CMOS bruges de altid parvis for at minimere strømforbrug.





Implementation af en inverter

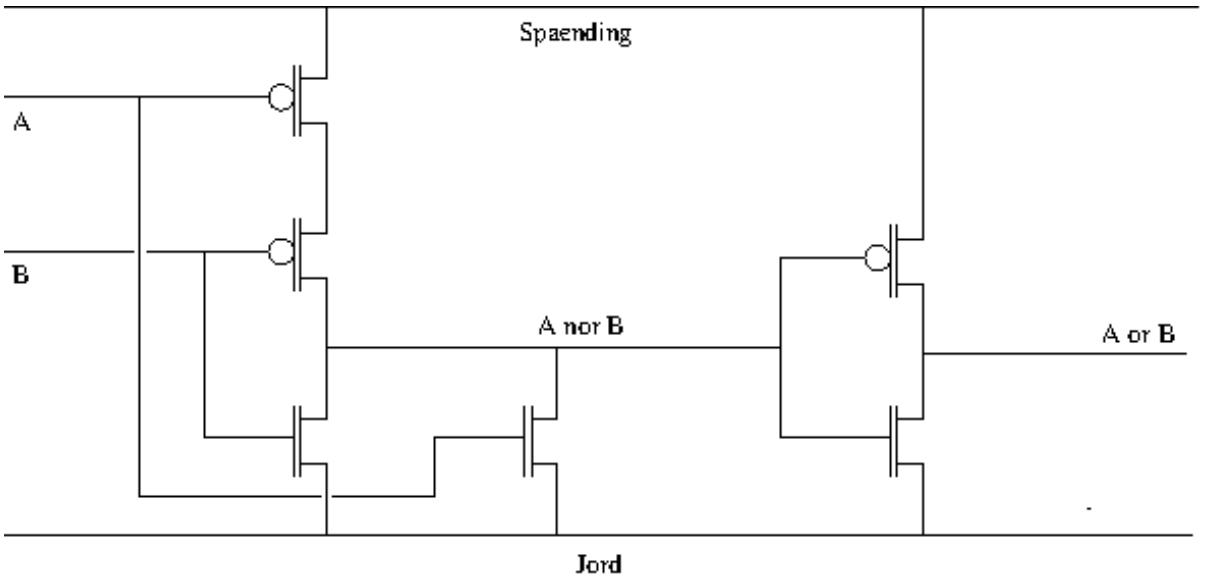
Den simpleste digitallogiske byggeklods:



I CMOS skal man altid bruge PMOS transistorer til at løfte spændingen (dvs forbinde direkte eller indirekte til strømforsyningen), mens NMOS skal bruges til at sænke spændingen (dvs forbinde direkte eller indirekte til "jord").

Implementation af en Or-port

Og en Nor-port.



En Or-port laves af en Nor-port med en inverter bagefter

Transistorer

Bedre end ventiler og vandslanger!

- Holdbare: Ingen bevægelige dele (bortset fra elektroner)
- Produceres ved en litografisk process.
- Kan laves ekstremt små
- Kan arbejde utroligt hurtigt (få pico-sekunder om at skifte, måske 5-20ps når de føder andre transistorer tæt på)
- i CMOS omsættes tilstrækkelig lidt energi til at man kan integrere *mange* transistorer på meget lidt plads (over en milliard på en kvadrat-centimeter)

Wikipedia har en fornuftig artikel, hvis du vil grave:
https://en.wikipedia.org/wiki/CMOS

Langt ude: http://visual6502.org/JSSim/expert-6800.html

Transistorer (II)

- Effekt er cirka proportional med skifte-frekvens, kvadratet på spændingen og kredsløbets kapacitans
- Vi kan påvirke skifte-frekvens på design tidspunktet
  - Design så ledninger skifter værdi så sjældent som muligt
  - F.eks. ved at stoppe for clock-signalet til de dele af chippen der ikke laver noget på et givent tidspunkt
- Maksimal skifte-frekvens er cirka proportional med spænding
- DVFS: Dynamic voltage-frequency scaling. Vi skruer løbende op og ned for både spænding og clock-frekvens efter behov.

Alle nyere chips bruger DVFS.

Produktionsproces

Sådan laver du en chip!

1. Reducer hele dit design til porte
2. Placer dine porte i et plan og forbind dem. Også kaldet "place and route". Der er kun ganske få adskildte lag hvori forbindelser kan løbe.
3. Beregn alle forsinkelser for signaler under en række forskellige scenarier (varians i ledningsevne, forsyningsspænding, skifte-aktivitet)
4. Beregn herfra max clock frekvens og strømforbrug
5. Hvis du er utilfreds, start forfra
6. Oversæt dit design til "masker" der kan styre en efterfølgende litografisk process
7. Producer.
8. Test og sorter. Sælg!

https://en.wikipedia.org/wiki/Semiconductor_device_fabrication

Moores lov

Moores lov er ikke en lov i samme forstand som tyngdeloven.

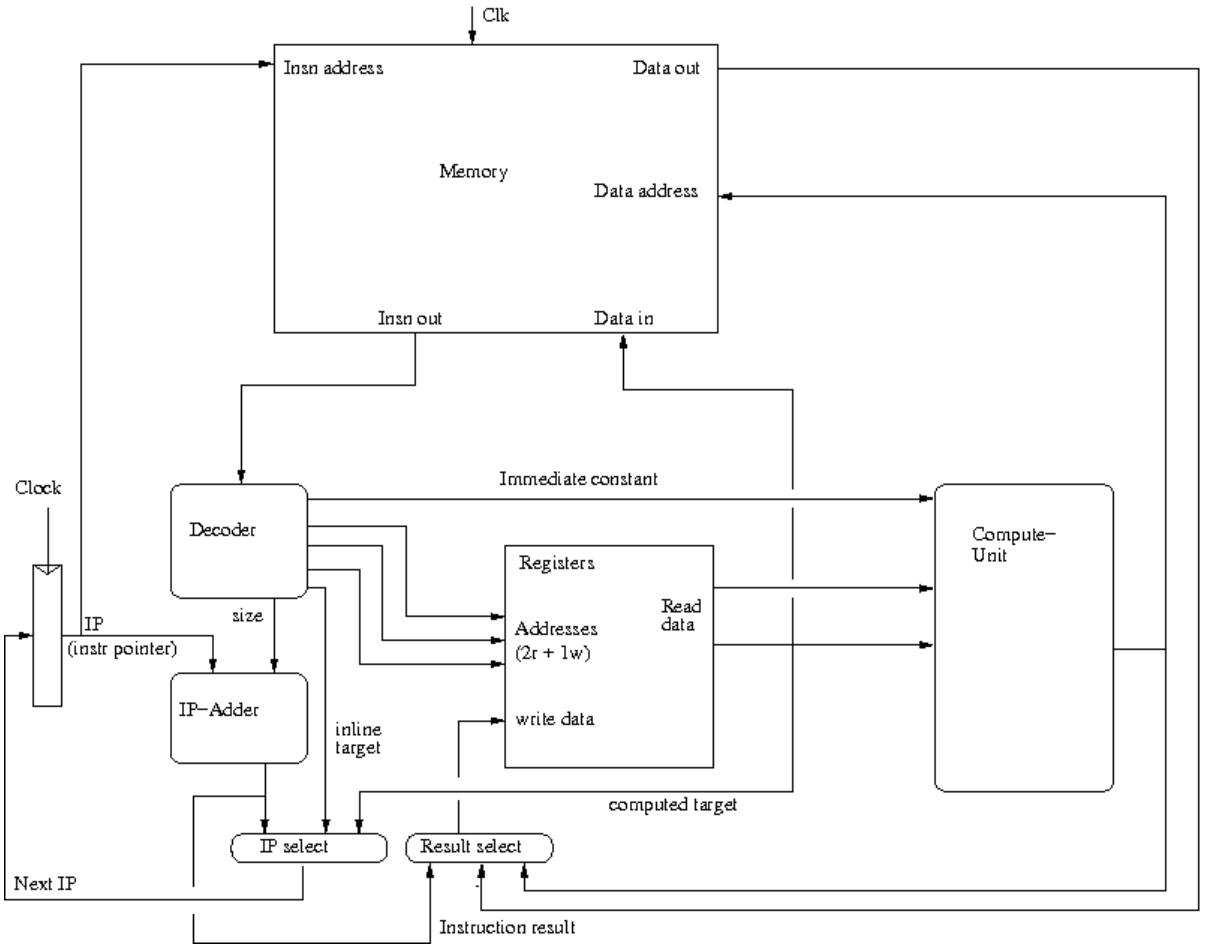
- Moores lov er et løfte. En afstemning af forventninger og mål for den industri der laver produktionsprocessen. Et løfte om løbende at sikre inkrementel forbedring af den litografiske process.
- Hver ny version af processen skal helst give transistorer som fylder halvt så meget som den foregående version
- Hver version kaldes iøvrigt en "process node" og angives ved størrelsen af den mindste detalje der kan laves.
- Vi har p.t. 14nm processen i brug, 10nm er ved at være der.
- Hver opgradering til næste process-version er dyr. En fabrik koster ca 30 milliarder.

Intel IDF 2014

Hvorfor hele tiden nye mikroarkitekturer?

- For at udnytte at flere og flere transistorer er til rådighed
- Eller at det bliver billigere og billigere at få produceret et kredsløb af fast kompleksitet
- Man kan ikke altid "bare" overføre en mikroarkitektur fra en fremstillingsprocess til en anden
    - Forholdet mellem transistorens styrke og kapacitansen på ledninger rykkes lidt ved hver ny og forbedret process version
    - Signal-skift bliver langsommere af at rejse langt eller drive andre transistorer
    - Langsomme signal-skift giver forøget energiforbrug
    - Kommunikation bliver dyrere relativt til beregning for hver ny og forbedret process version

A2 mikroarkitektur - revisited



Spørgsmål og Svar