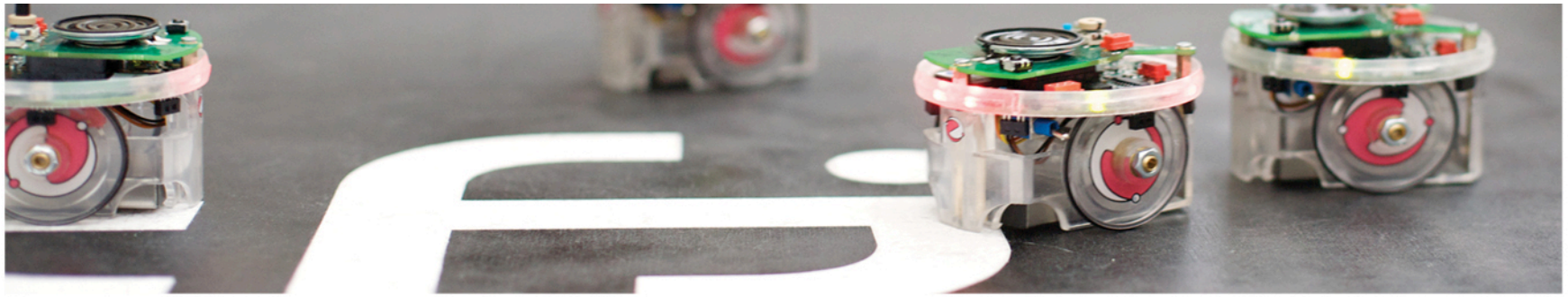


UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

INF1400

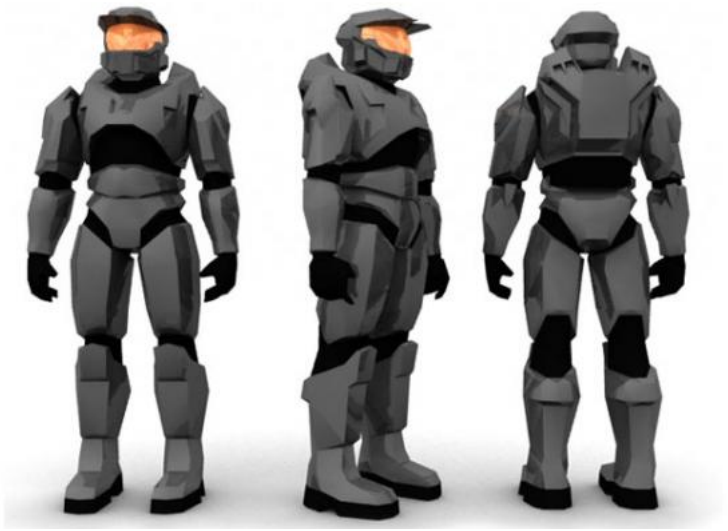
Maskinarkitektur

Hovedpunkter

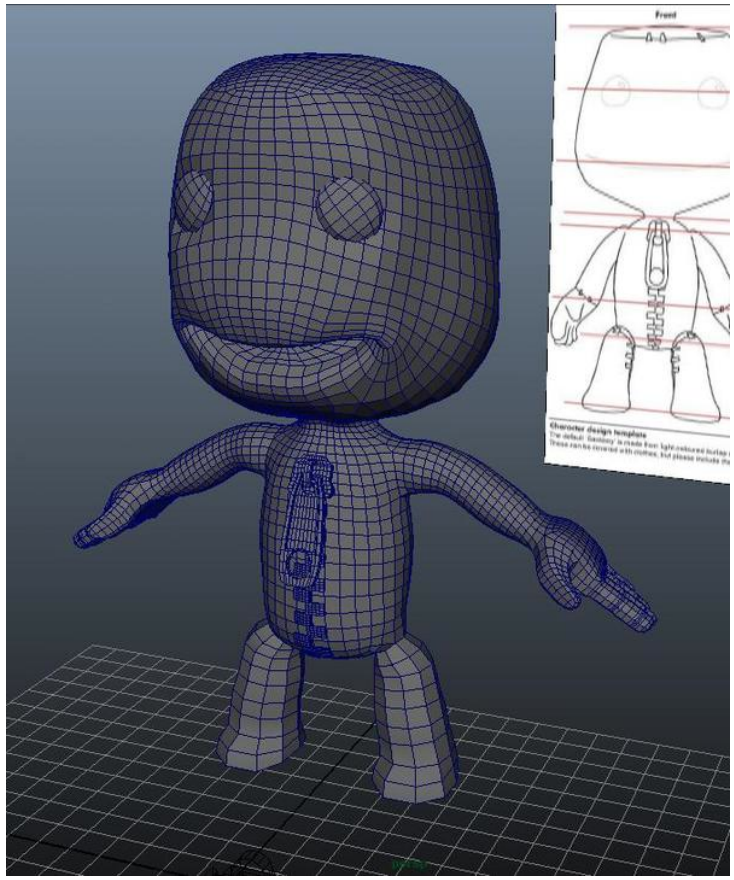
- Veien fra tastatur til bit
- Datamaskinarkitekturer
  - Datapath
  - Pipelining
- Parallell og multikjerner CPU
- Minne og Cache

Veien fra tastatur til en bit:

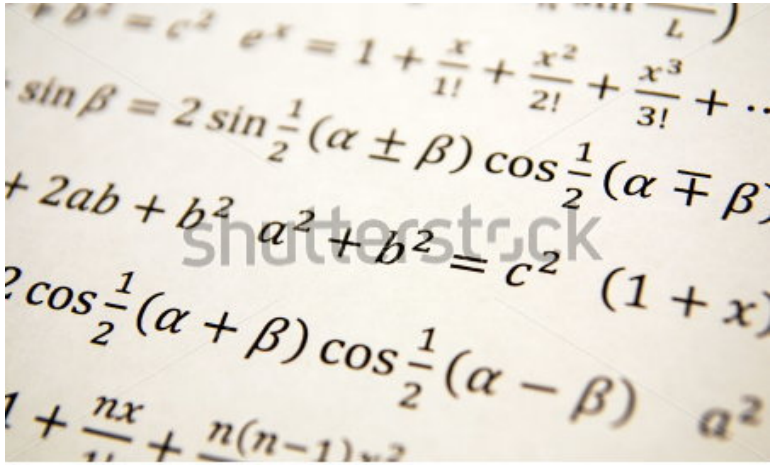
Bevegbar karakter i 3D



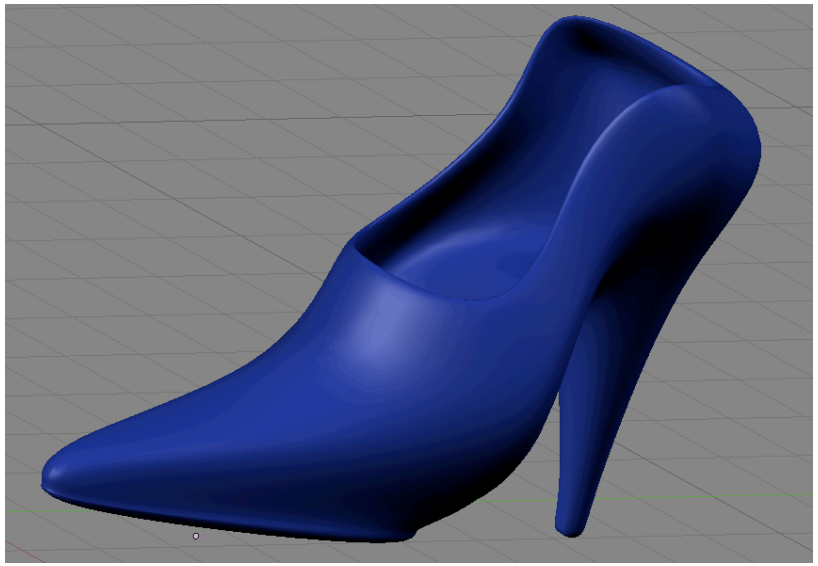
Karakteren tegnet i 3D program

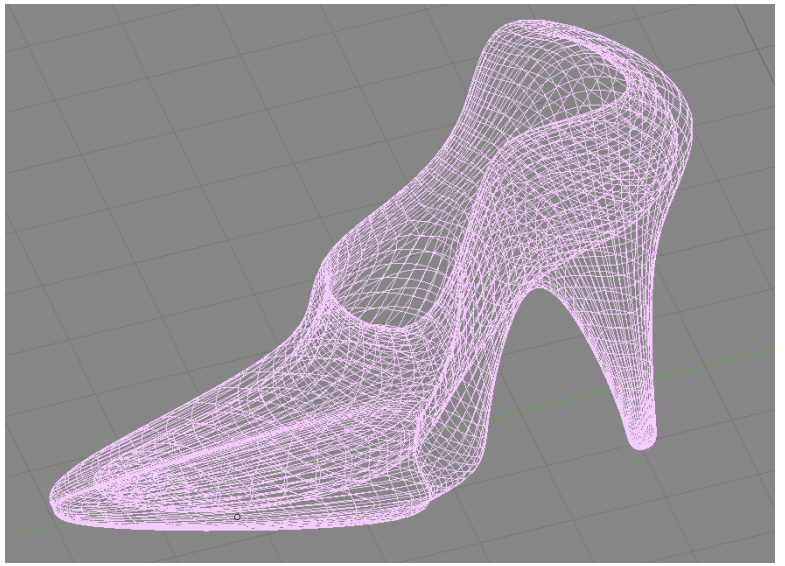


Karakteren består av MANGE polygoner



Bakenfor alle disse polygonene ligger det mye matematikk



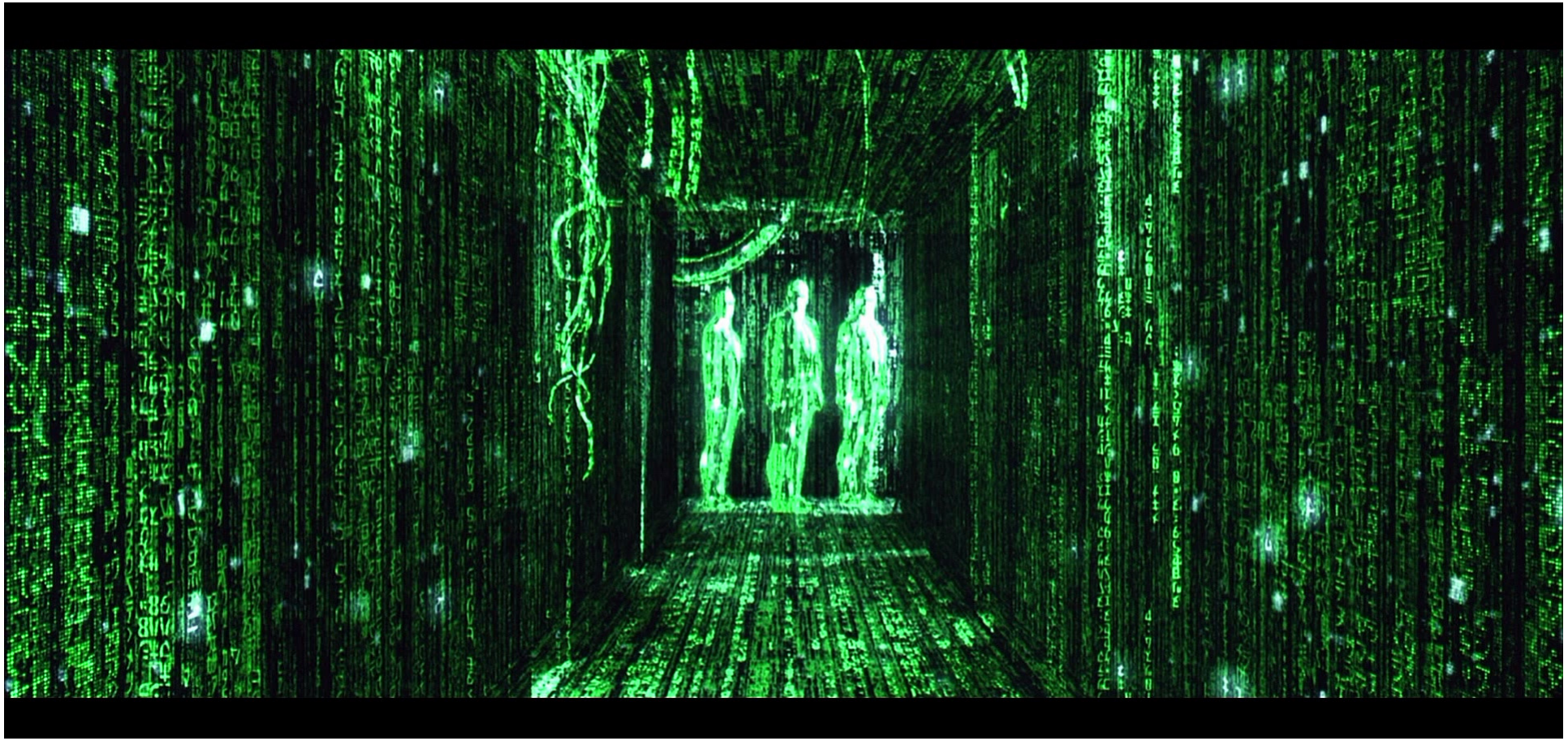


```
181    public void actionPerformed(ActionEvent e)
182    {
183        if (e.getSource() == forwardJButton)
184        {
185            Transform3D temp = new Transform3D();
186            viewObjectFromGroup.getTransform(temp);
187            Transform3D tempDelta = new Transform3D();
188            tempDelta.setTranslation(new Vector3f(0.0f, 0.0f, -1.0f));
189            temp.mul(tempDelta);
190            System.out.println(temp);
191            float matrix[] = new float[16]; //declare array of 16 floats for matrix
192            temp.get(matrix);
193            if (matrix[11] <= 1.0)//object front face z = 1
194            {
195                System.out.println("Don't multiply Transform3D at: "+matrix[11]);
196            }
197            else    //setTransform
198            {
199                viewObjectFromGroup.setTransform(temp);
200            }
201        }
```

All matematikk og grafikk blir kodet i et programmeringsspråk (eks. JAVA). Typiskt ligger en 3D spill på over 1 million linjer med kode

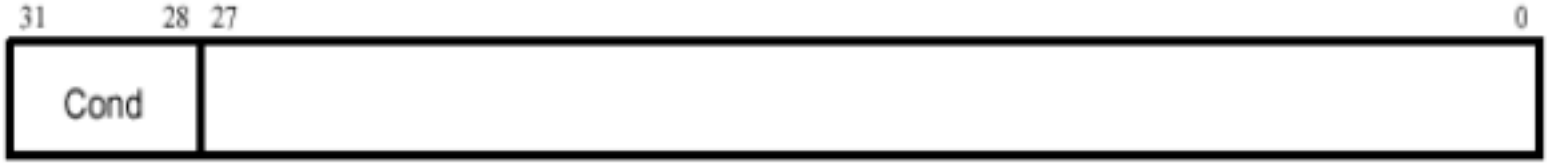
```
push    ebp
mov     ebp, esp
movzx   ecx, [ebp+arg_0]
pop     ebp
movzx   dx, cl
lea     eax, [edx+edx]
add     eax, edx
shl     eax, 2
add     eax, edx
shr     eax, 8
sub     cl, al
shr     cl, 1
add     al, cl
shr     al, 5
movzx   eax, al
retn
```

All kode blir kompilert til assembler (maskin nært programmeringsspråk). Det er også mulig å programmere rett på assembler, men ikke så enkelt og trivielt som høyere nivå programspråk.



Maskinkode ?

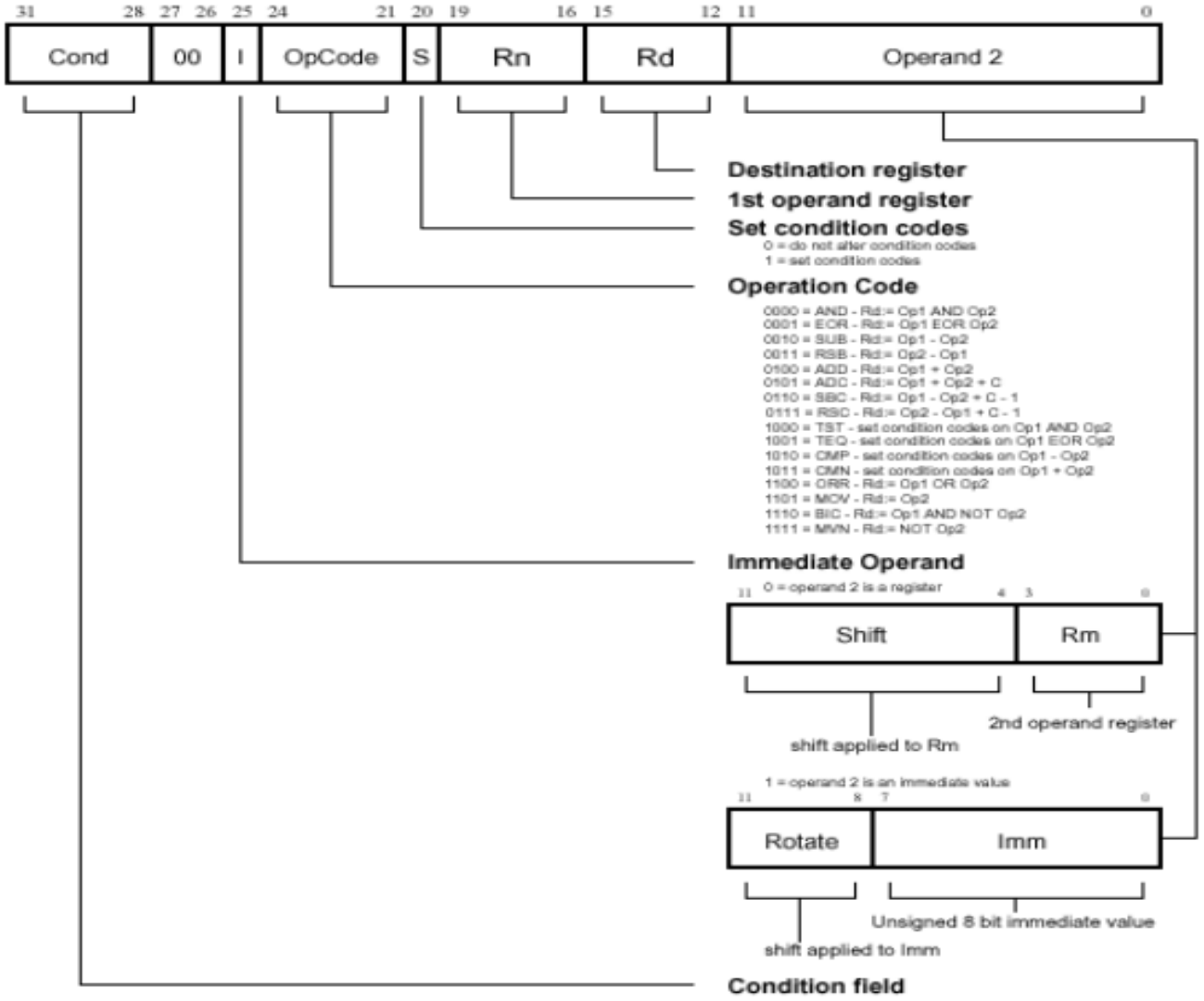
All assembly kode blir så kodet om til maskinkode. Dette er den koden maskinen forstår og utfører sekvensielt.

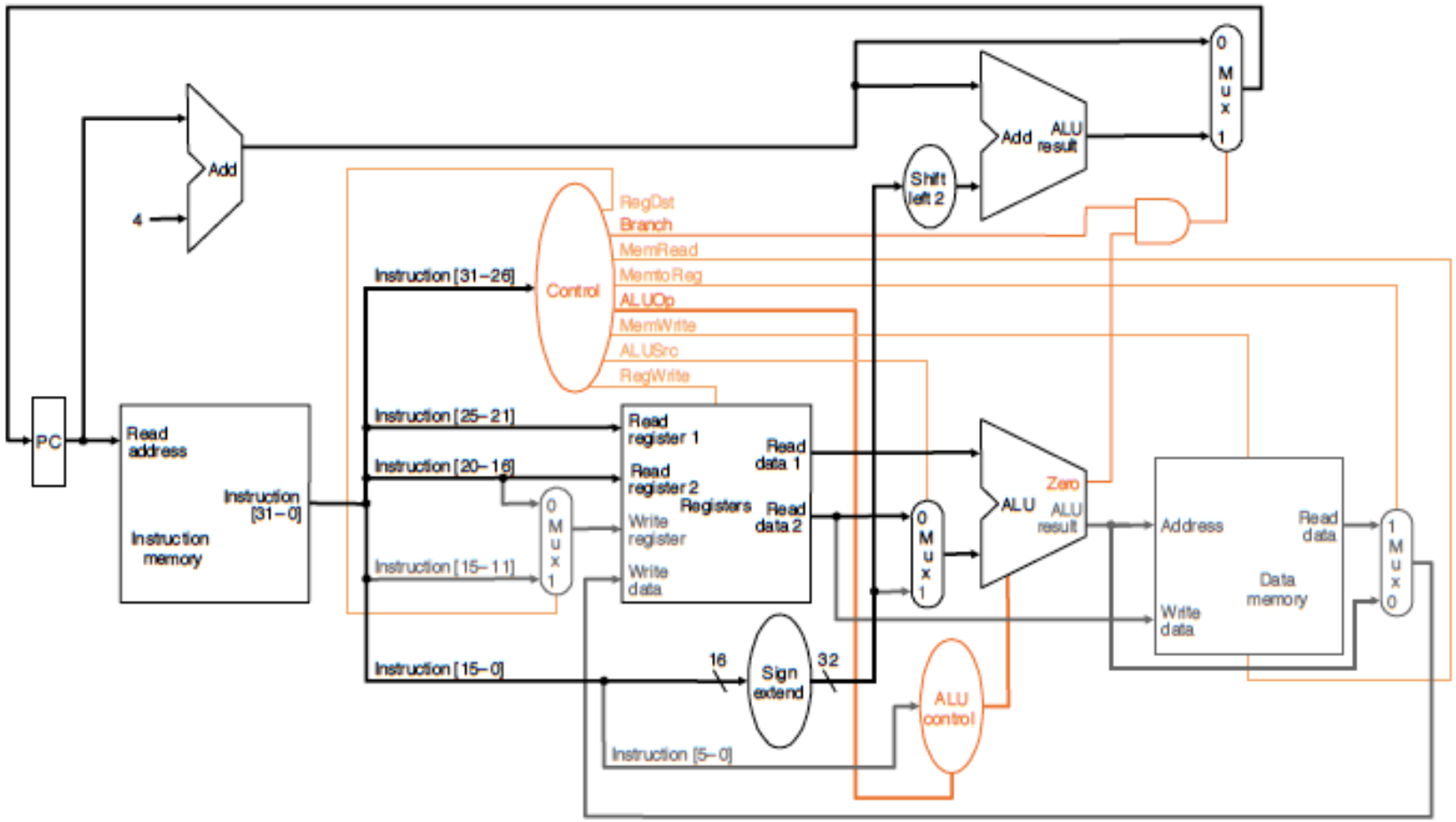


Condition field

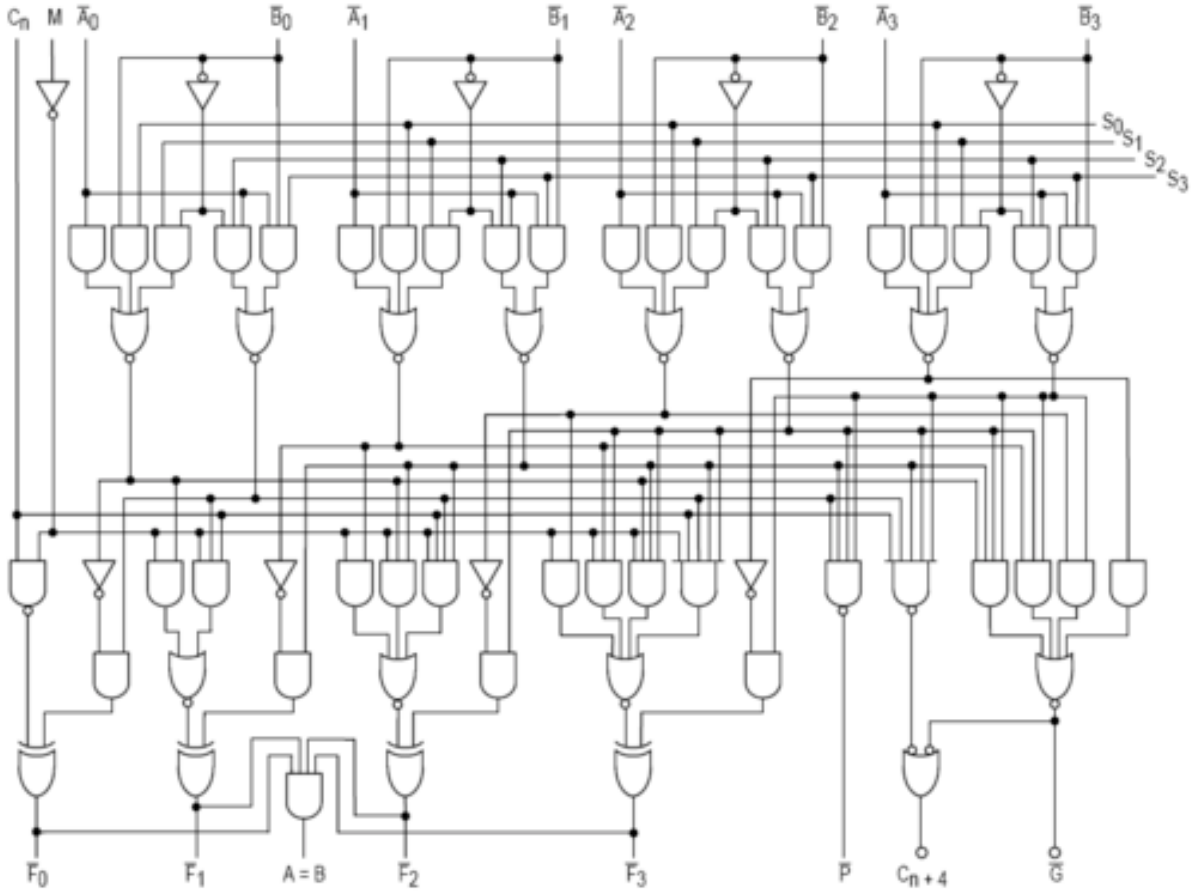
0000 = EQ - Z set (equal)
0001 = NE - Z clear (not equal)
0010 = CS - C set (unsigned higher or same)
0011 = CC - C clear (unsigned lower)
0100 = MI - N set (negative)
0101 = PL - N clear (positive or zero)
0110 = VS - V set (overflow)
0111 = VC - V clear (no overflow)
1000 = HI - C set and Z clear (unsigned higher)
1001 = LS - C clear or Z set (unsigned lower or same)
1010 = GE - N set and V set, or N clear and V clear (greater or equal)
1011 = LT - N set and V clear, or N clear and V set (less than)
1100 = GT - Z clear, and either N set and V set, or N clear and V clear (greater than)
1101 = LE - Z set, or N set and V clear, or N clear and V set (less than or equal)
1110 = AL - always
1111 = NV - never

Instruksjonsbit for en 32-bits arkitektur

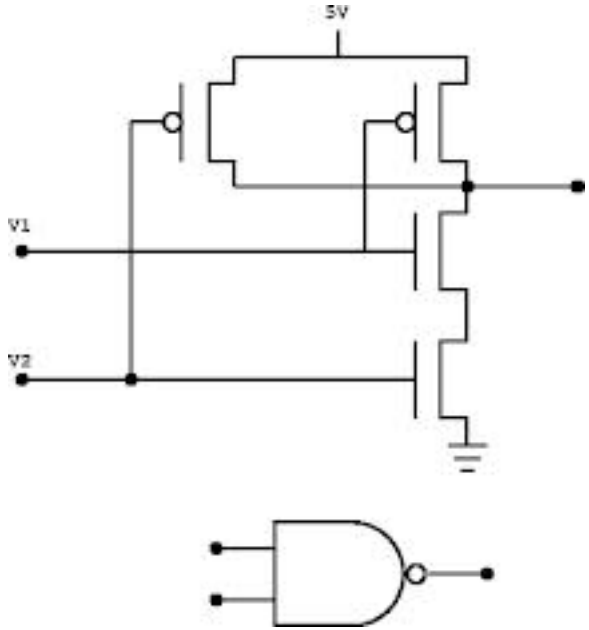

31 28 27 26 25 24 21 20 19 16 15 12 11 0
Cond
00
I
OpCode
S
Rn
Rd
Operand 2
Destination register
1st operand register
Set condition codes
0 = do not alter condition codes
1 = set condition codes
Operation Code
0000 = AND - Rd:= Op1 AND Op2
0001 = EOR - Rd:= Op1 EOR Op2
0010 = SUB - Rd:= Op1 - Op2
0011 = RSB - Rd:= Op2 - Op1
0100 = ADD - Rd:= Op1 + Op2
0101 = ADC - Rd:= Op1 + Op2 + C
0110 = SBC - Rd:= Op1 - Op2 + C - 1
0111 = RSC - Rd:= Op2 - Op1 + C - 1
1000 = TST - set condition codes on Op1 AND Op2
1001 = TEQ - set condition codes on Op1 EOR Op2
1010 = CMP - set condition codes on Op1 - Op2
1011 = CMN - set condition codes on Op1 + Op2
1100 = ORR - Rd:= Op1 OR Op2
1101 = MOV - Rd:= Op2
1110 = BIC - Rd:= Op1 AND NOT Op2
1111 = MVN - Rd:= NOT Op2
Immediate Operand
0 = operand 2 is a register
11 4 3 0
Shift
Rm
2nd operand register
shift applied to Rm
1 = operand 2 is an immediate value
11 8 7 0
Rotate
Imm
Unsigned 8 bit immediate value
shift applied to Imm
Condition field



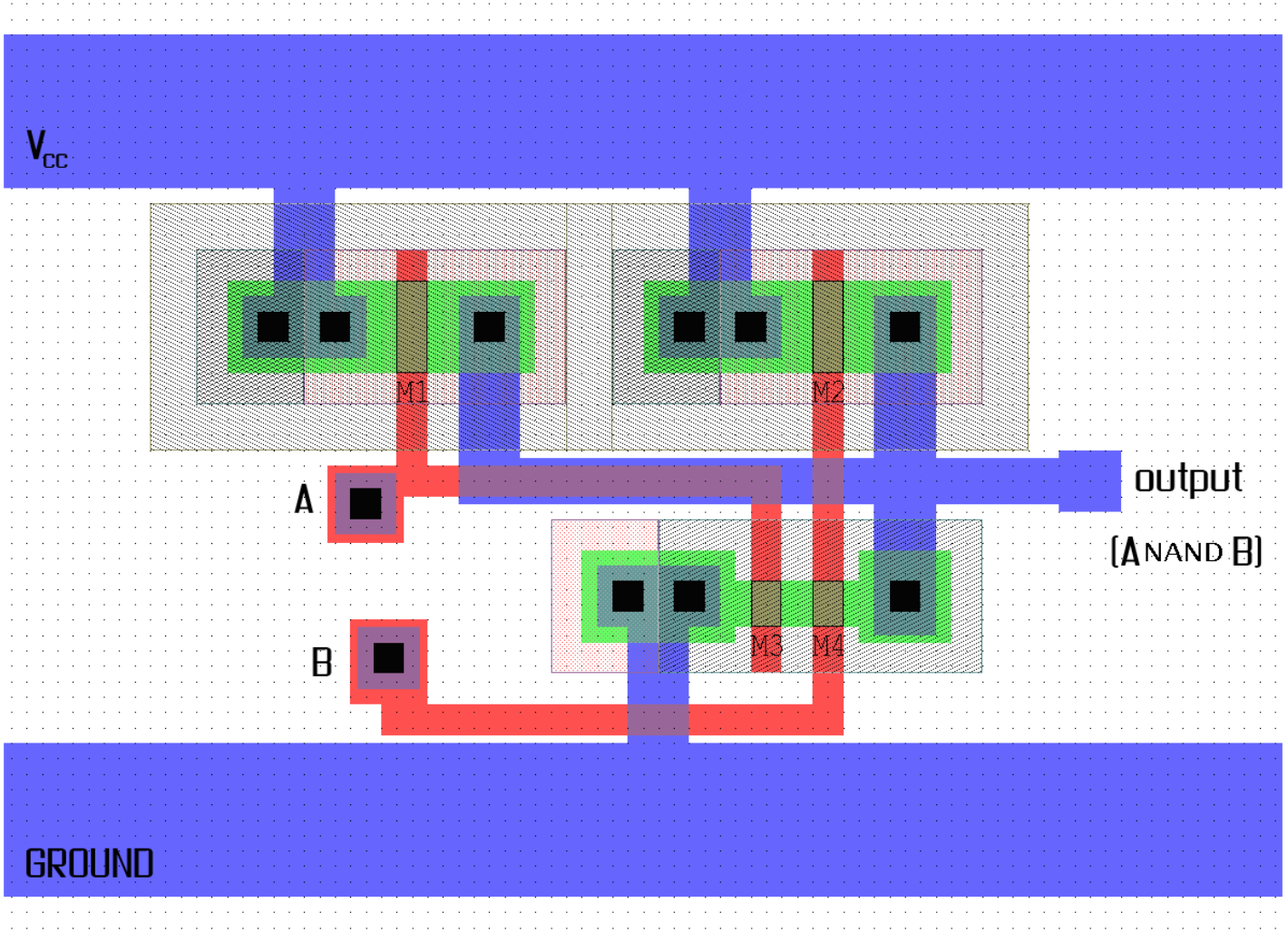
Et eksempel for en 32-bits datapath



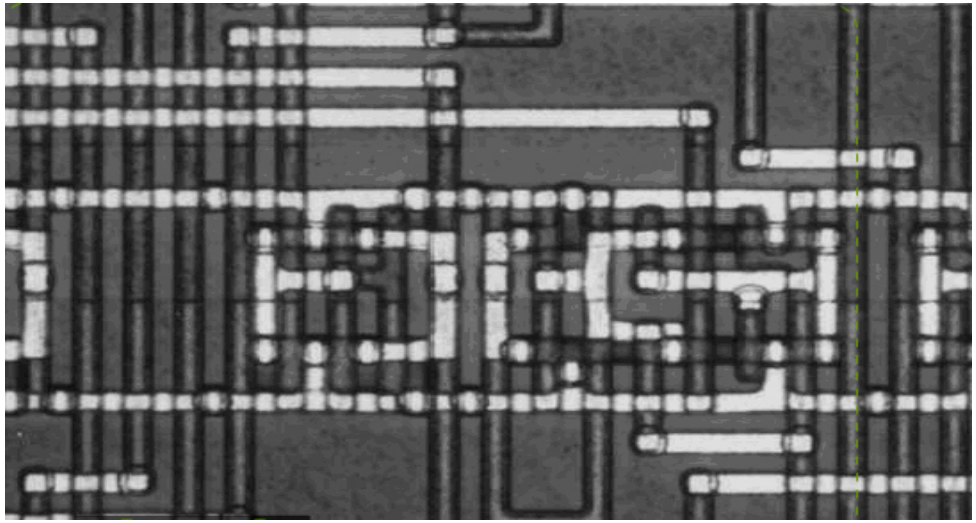
Et eksempel på en ALU

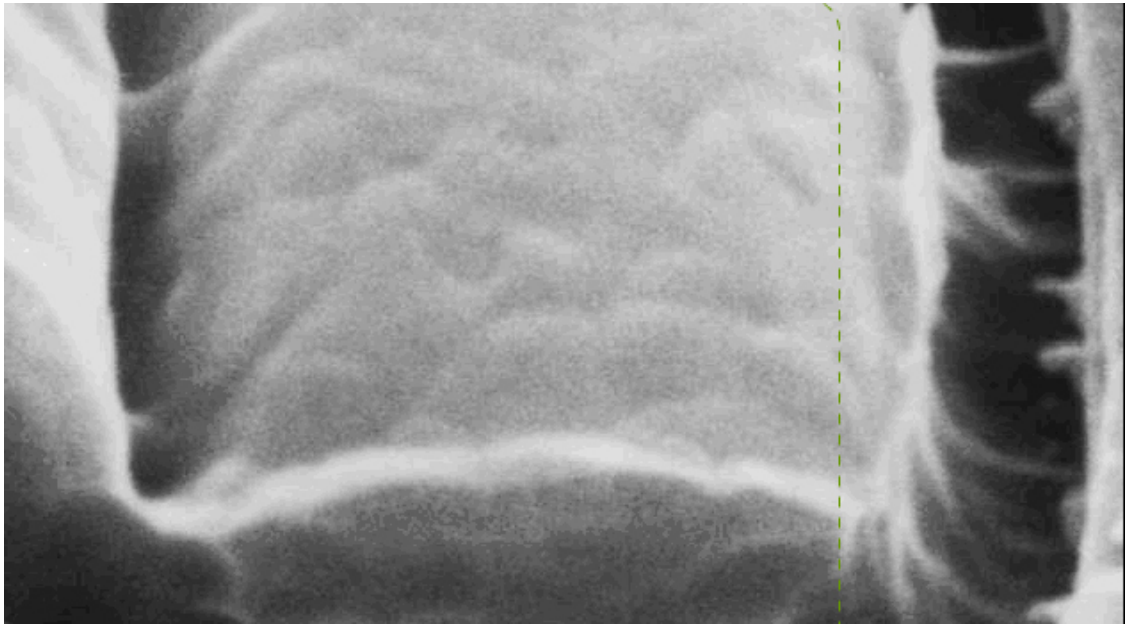


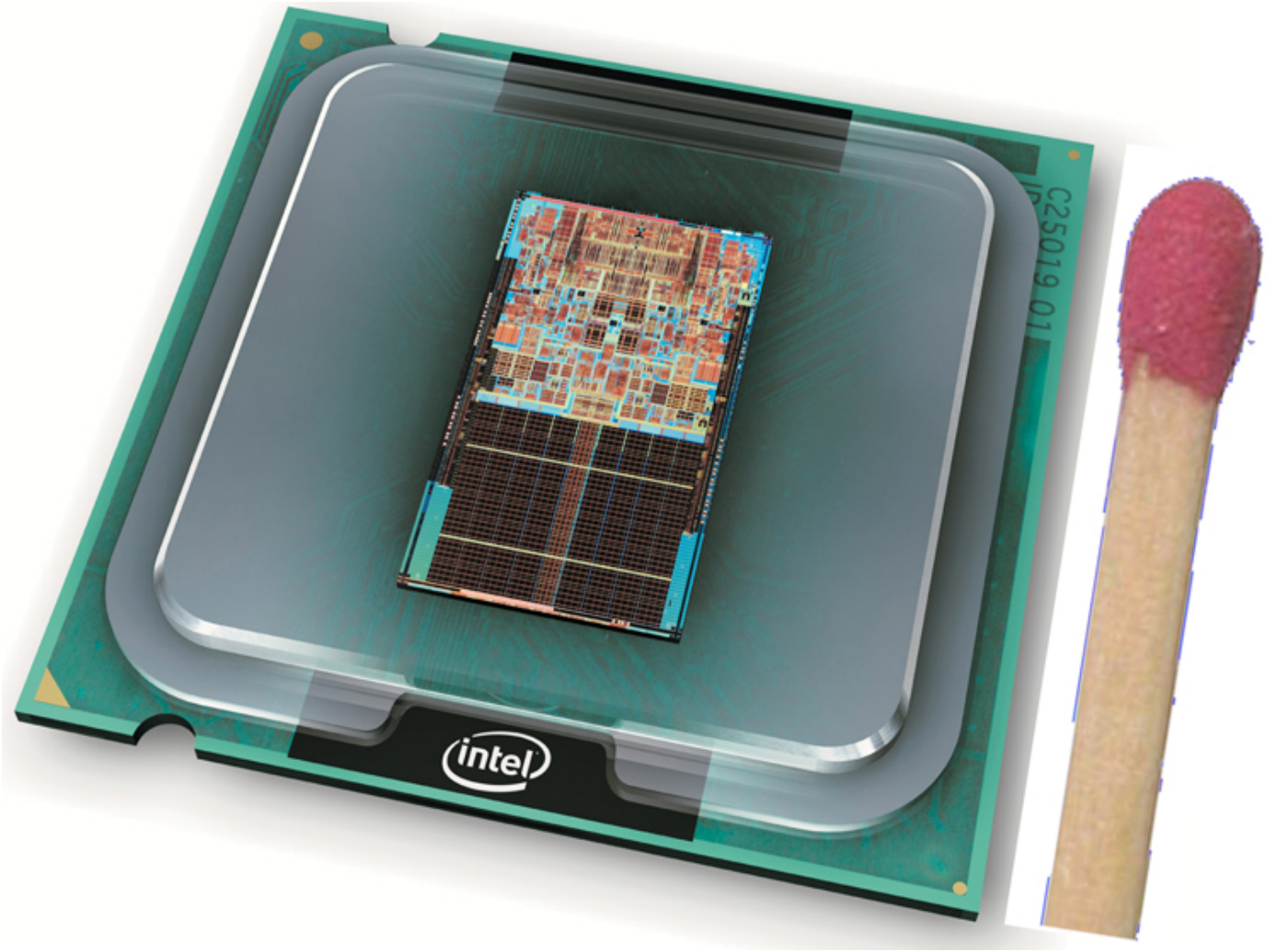
Nand-gate på transistor nivå



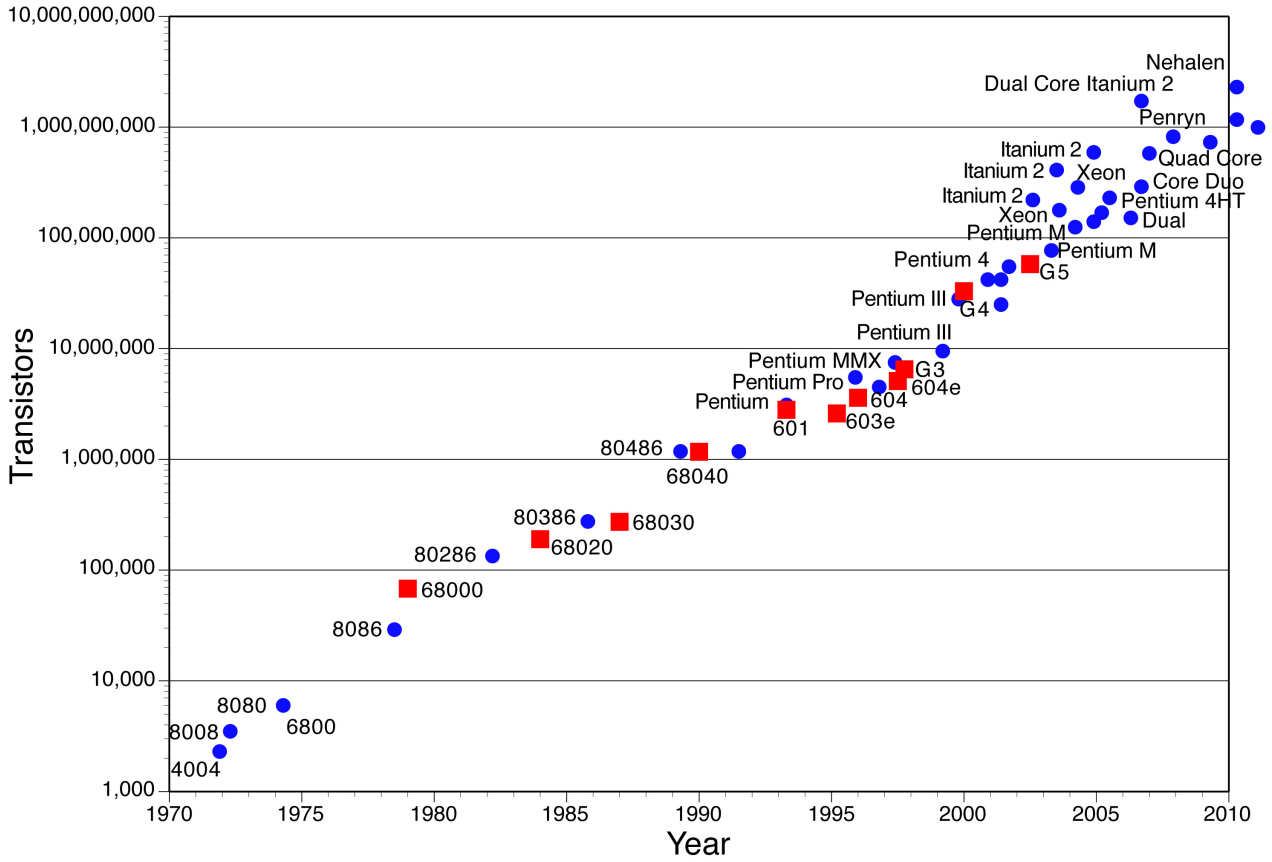
Nand-gate på Layout skjema





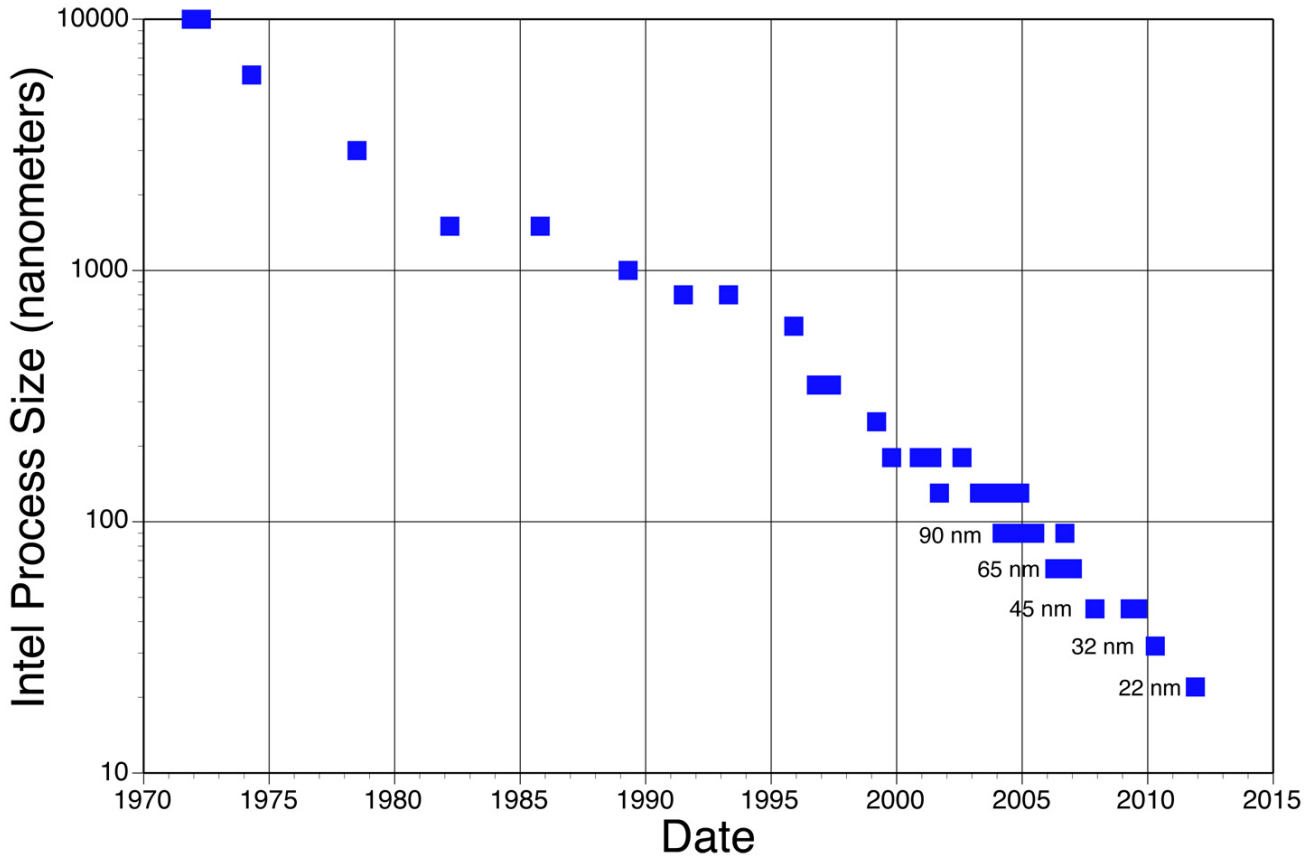

intel

Teknologi utvikling

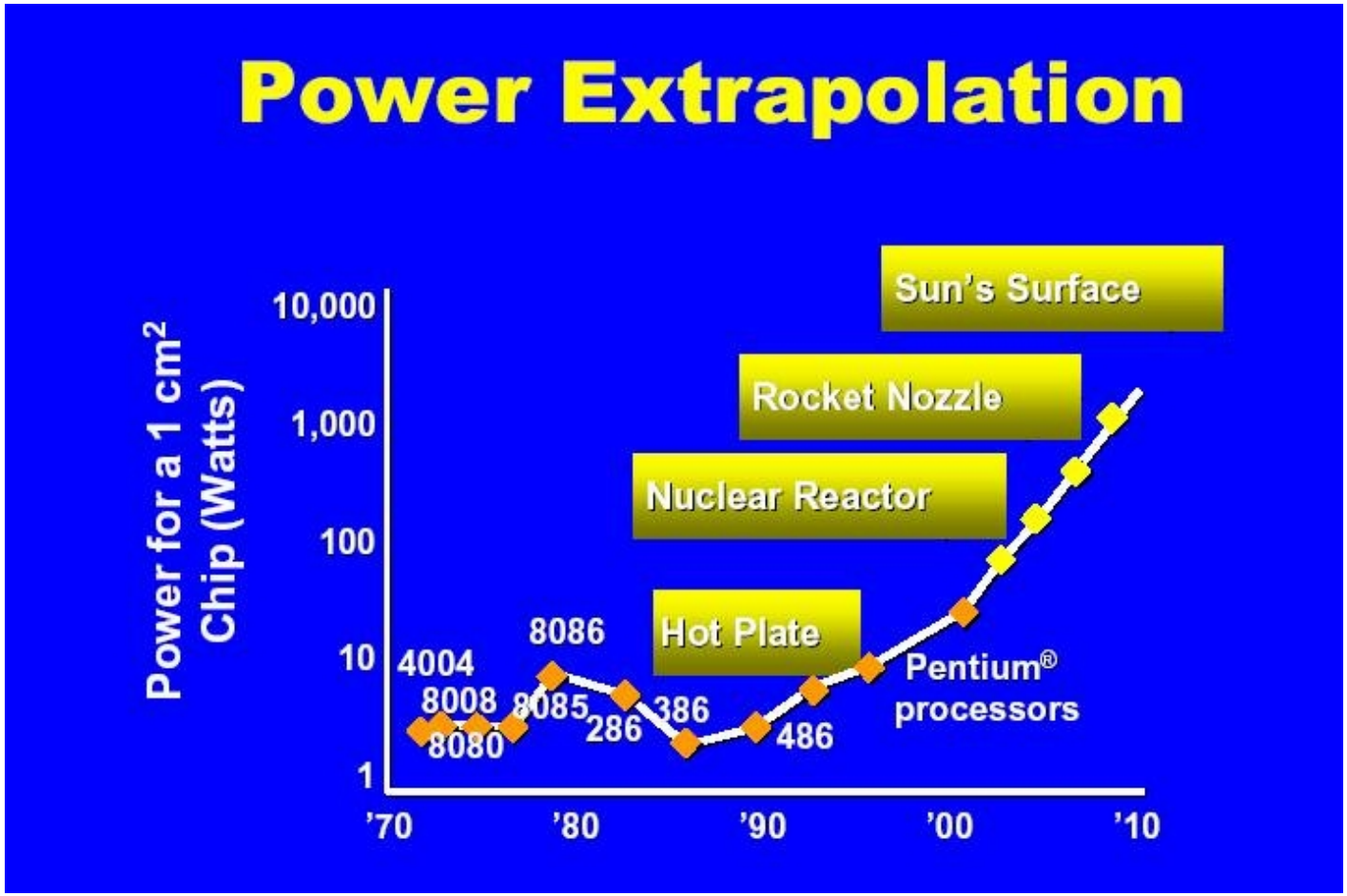


© mrsec.wisc.edu/

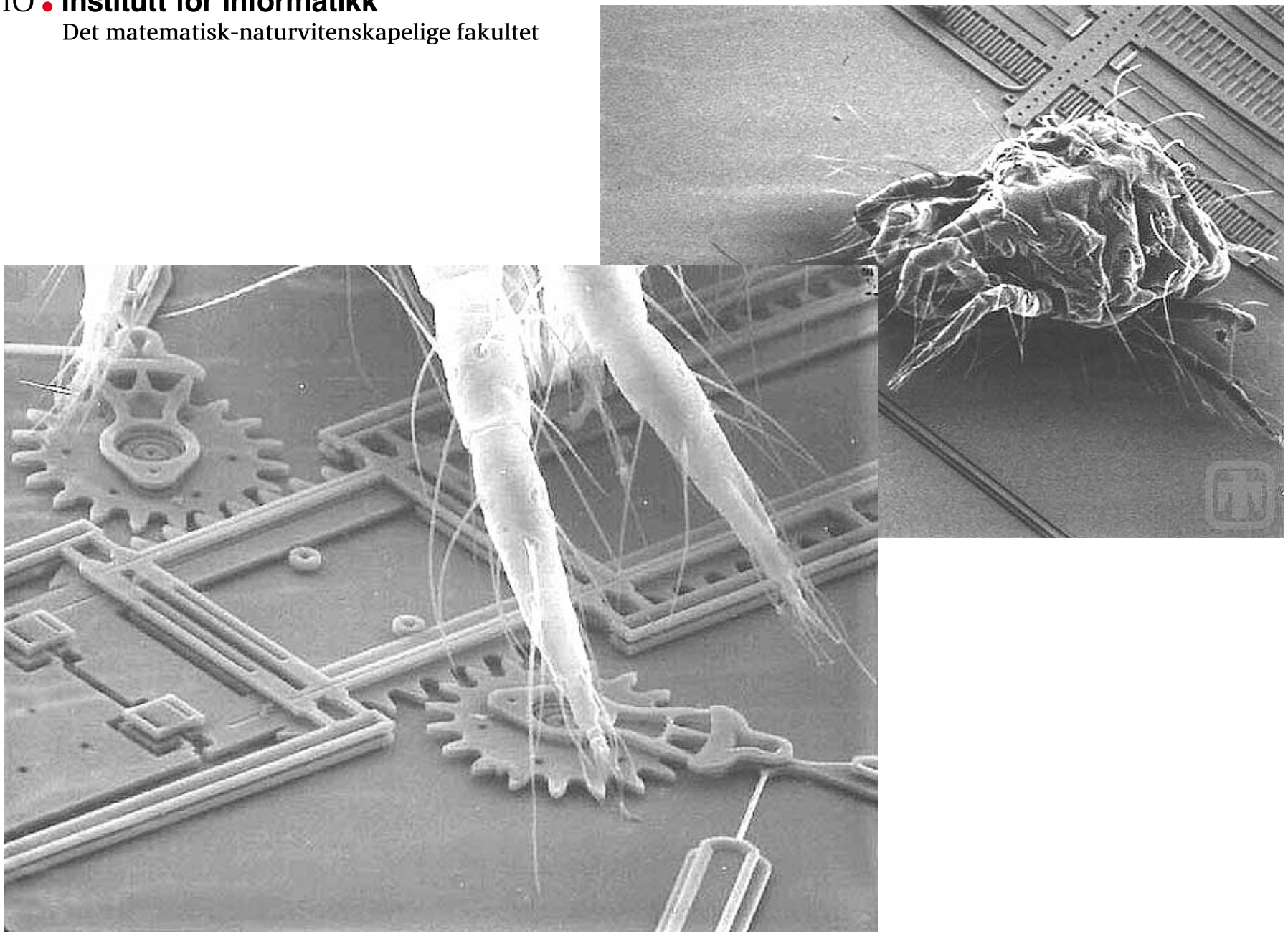
Prosessutvikling



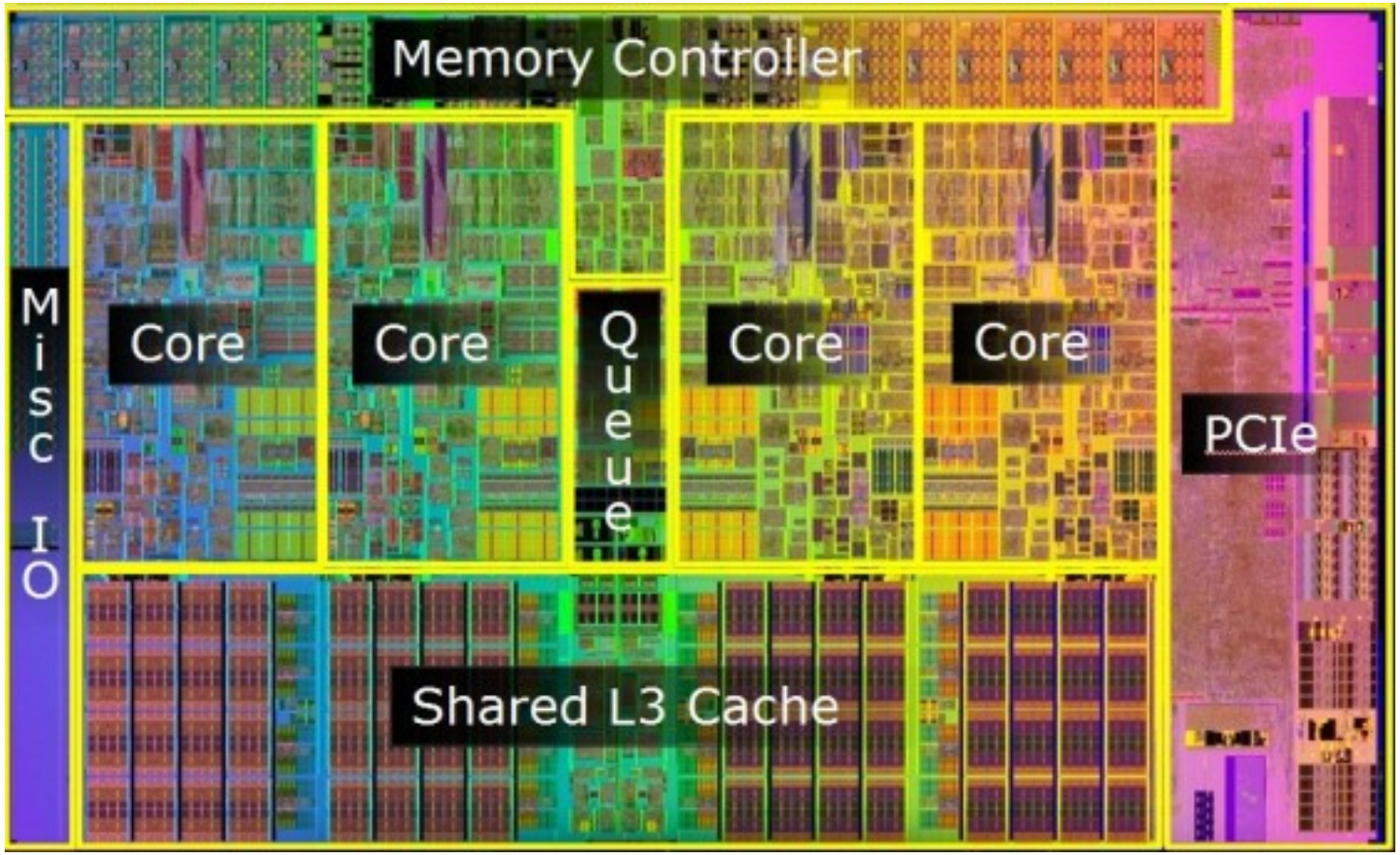
Global oppvarming?



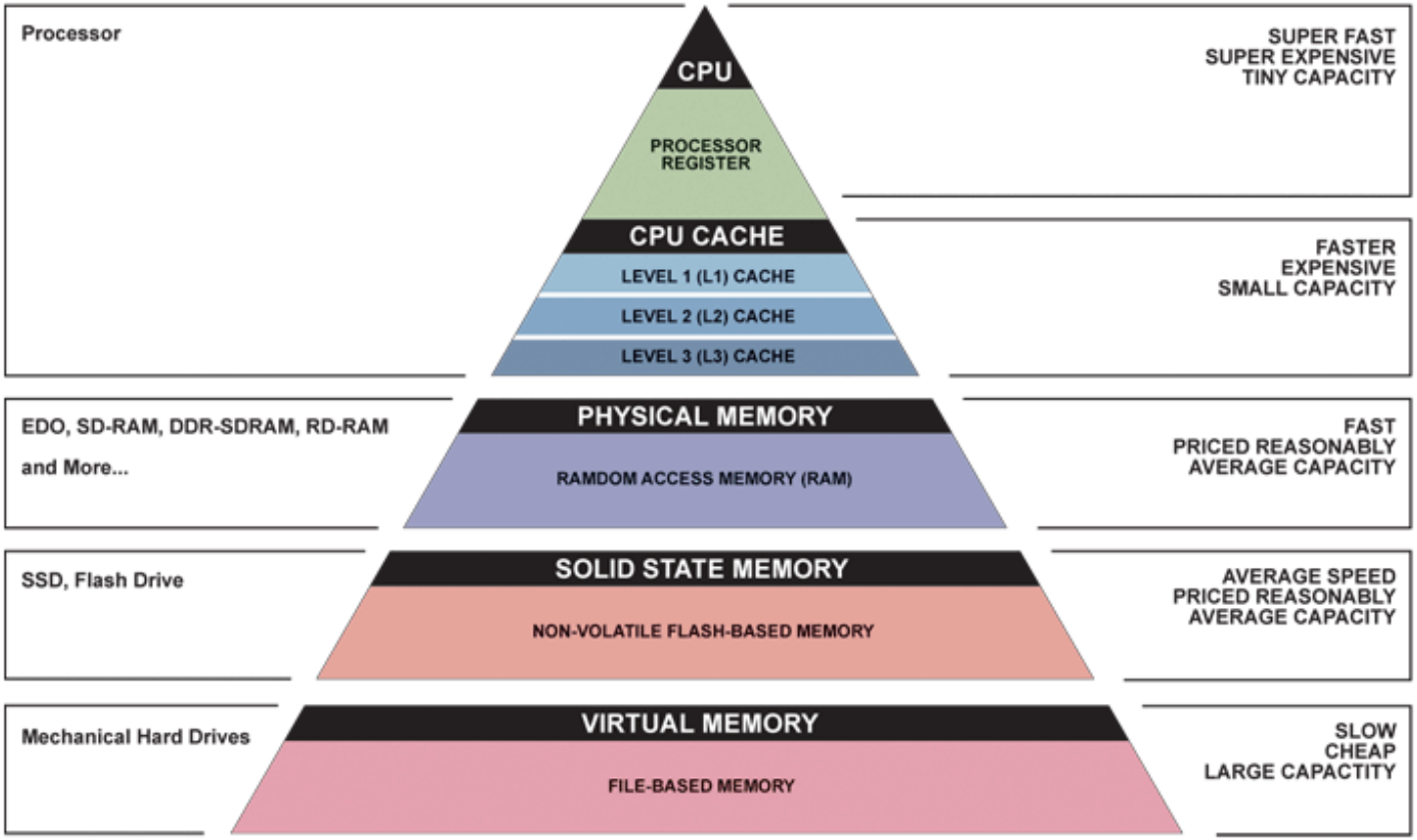
UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet



Intel Core i7



Minne


Processor
CPU
PROCESSOR REGISTER
CPU CACHE
LEVEL 1 (L1) CACHE
LEVEL 2 (L2) CACHE
LEVEL 3 (L3) CACHE
SUPER FAST
SUPER EXPENSIVE
TINY CAPACITY
FASTER
EXPENSIVE
SMALL CAPACITY
EDO, SD-RAM, DDR-SDRAM, RD-RAM
and More...
PHYSICAL MEMORY
RAMDOM ACCESS MEMORY (RAM)
FAST
PRICED REASONABLY
AVERAGE CAPACITY
SSD, Flash Drive
SOLID STATE MEMORY
NON-VOLATILE FLASH-BASED MEMORY
AVERAGE SPEED
PRICED REASONABLY
AVERAGE CAPACITY
Mechanical Hard Drives
VIRTUAL MEMORY
FILE-BASED MEMORY
SLOW
CHEAP
LARGE CAPACTITY

Datamaskinarkitektur

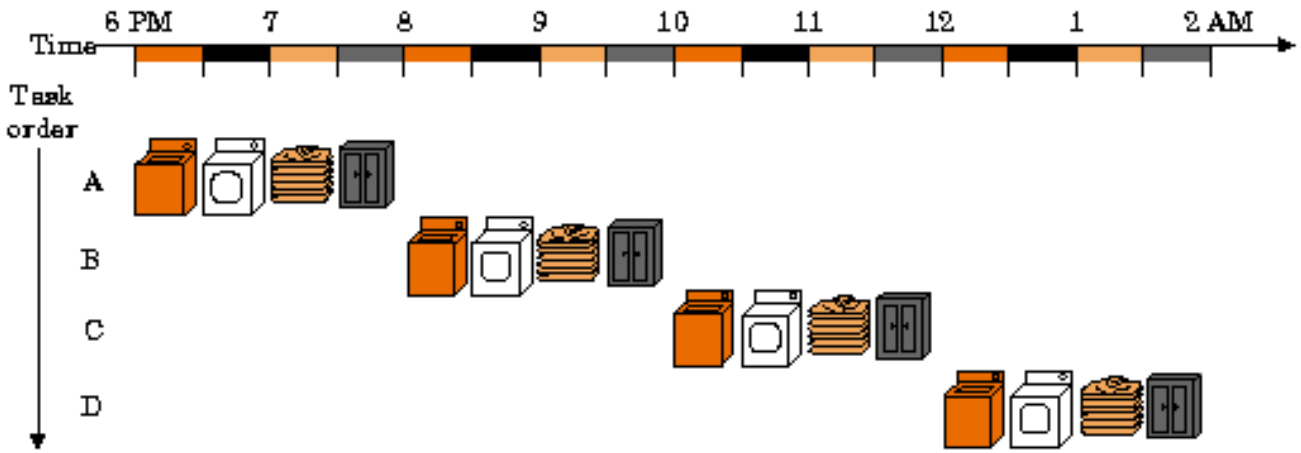
Pipelining

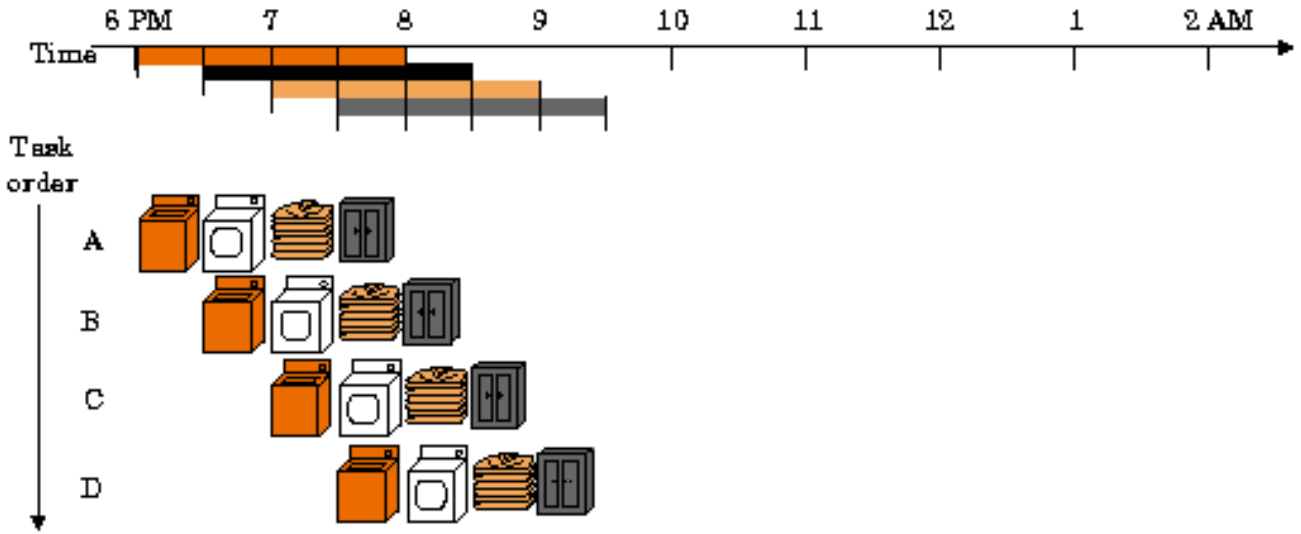
- Innfører samlebåndsprinsipp for eksekvering av instruksjoner
- Hver instruksjon må splittes opp i uavhengige deler (subinstruksjoner) som utføres etter hverandre
- Hver subinstruksjon kan utføres uavhengig av de andre subinstruksjonene
- Neste instruksjon settes igang før forrige instruksjon er helt ferdig.
- **NB!! Hver instruksjon tar like lang tid å utføre, men prosessoren utfører flere instruksjoner i et gitt tidsrom!**

Pipelining - Analogi

- Vasking av klær

1. Vaske i vaskemaskin
2. Trøke i tørketrommel
3. Brette sammen
4. Sette dem i skapet

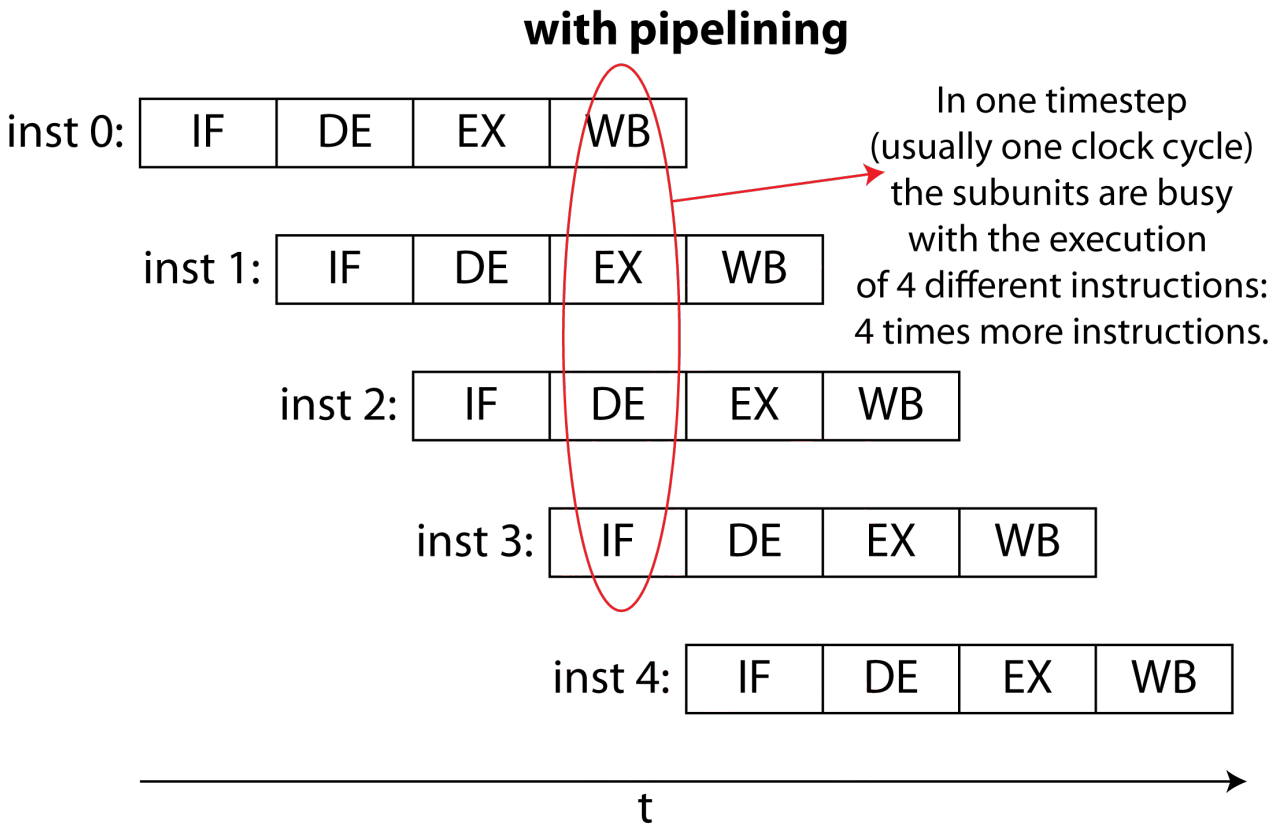

6 PM
7
8
9
10
11
12
1
2 AM
Time
Task order
A
B
C
D

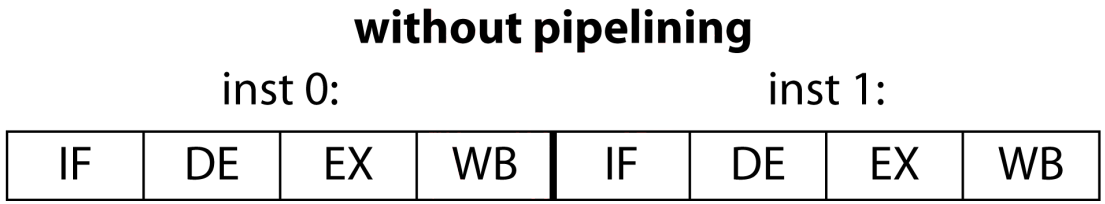

6 PM
7
8
9
10
11
12
1
2 AM
Time
Task order
A
B
C
D

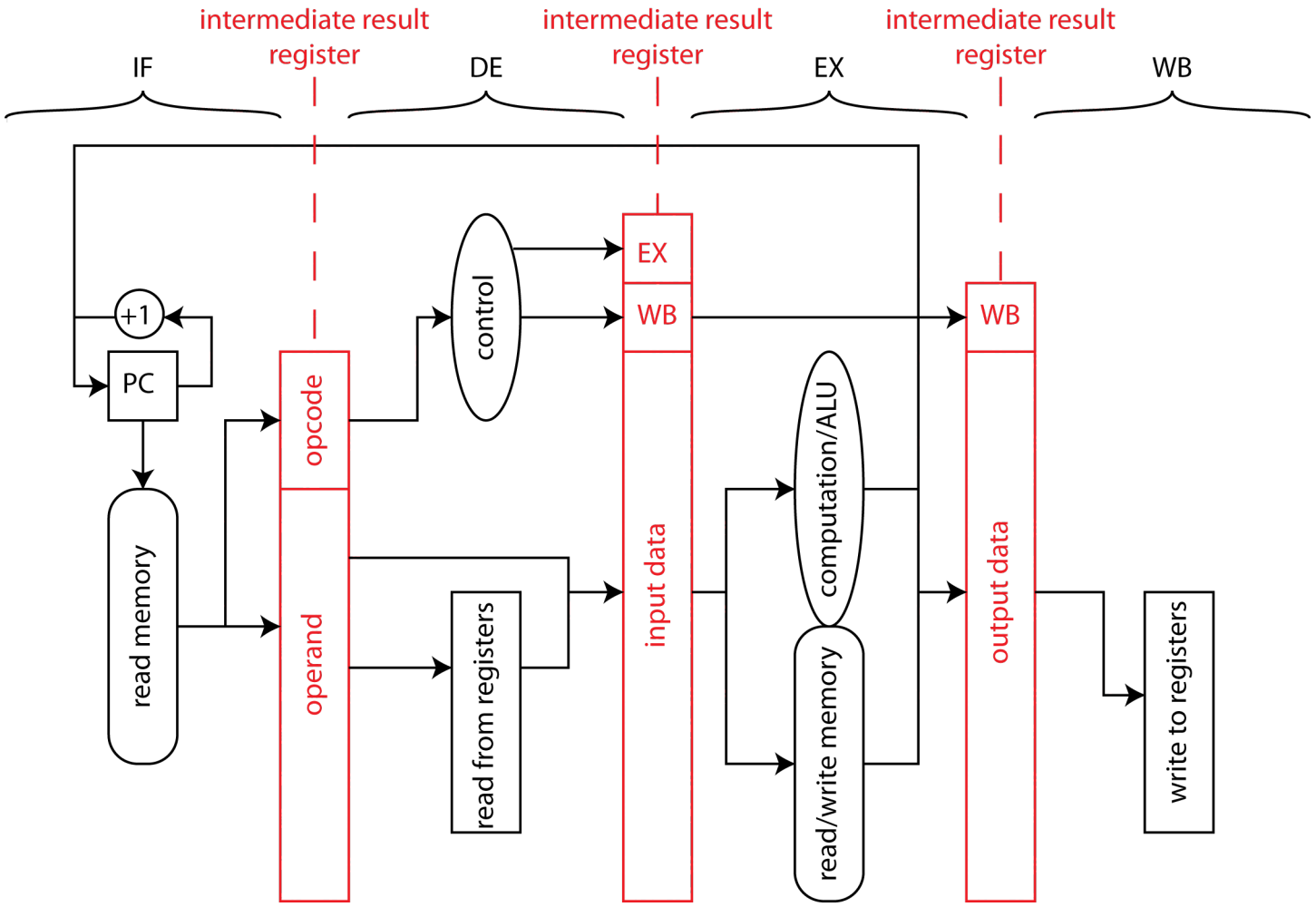
Pipelining instruksjonssett

- Eksempelvis kan vi ha følgende subinstruksjoner i en pipeline
  - IF: Instruction fetch (get the instruction)
  - DE: decode and load (from a register)
  - EX: Execute
  - WB: Write back (write the result to a register)

- **PS! Read og write fra register/MEM kan gjøres i hver sin halvdel av en sykel (1/2 klokkeperiode)**


with pipelining
inst 0:
IF
DE
EX
WB
inst 1:
IF
DE
EX
WB
inst 2:
IF
DE
EX
WB
inst 3:
IF
DE
EX
WB
inst 4:
IF
DE
EX
WB
In one timestep
(usually one clock cycle)
the subunits are busy
with the execution
of 4 different instructions:
4 times more instructions.
t


without pipelining
inst 0:
inst 1:
IF
DE
EX
WB
IF
DE
EX
WB


intermediate result register
intermediate result register
intermediate result register
IF
DE
EX
WB
+1
PC
read memory
opcode
operand
control
read from registers
EX
WB
input data
computation/ALU
read/write memory
WB
output data
write to registers

Pipeline med flere trinn

- Denne 4-trinns pipeline er den korteste som finnes for en CPU.
- Moderne CPU bruker vesentlig flere trinn
- Pentium III har 16 trinn, Pentrium 4 har 31 trinn

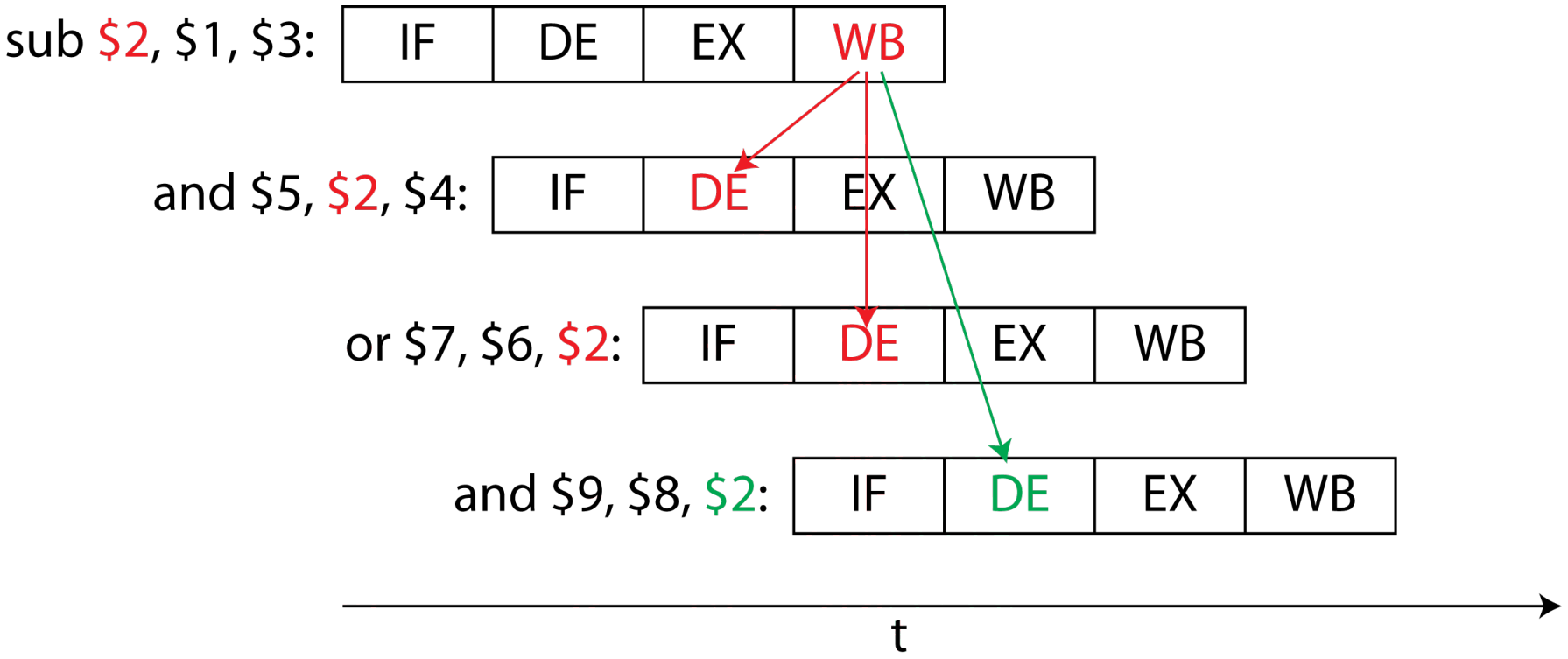
Speed-up

- Det å ha en 4 trinns pipeline betyr ikke at man får 4 ganger raskere prosessering. Det går alltid noe tid bort til administrering av instruksjoner.

Komplikasjoner ved pipelining

- Til enhver tid kan det være subinstruksjoner fra opptil 4 instruksjoner i en pipeline
- Noen ganger er ikke alle subinstruksjonene gyldige
- Neste instruksjon kan ikke eksekveres rett etter hvis hopp-betingelsen slår til
- En slik situasjon kalles HAZARD. Vi har tre typer:
    1. Resource hazard
    2. Data hazard
    3. Control Hazard

Data Hazard


sub $2, $1, $3:
IF
DE
EX
WB
and $5, $2, $4:
IF
DE
EX
WB
or $7, $6, $2:
IF
DE
EX
WB
and $9, $8, $2:
IF
DE
EX
WB
t