Computerarkitektur

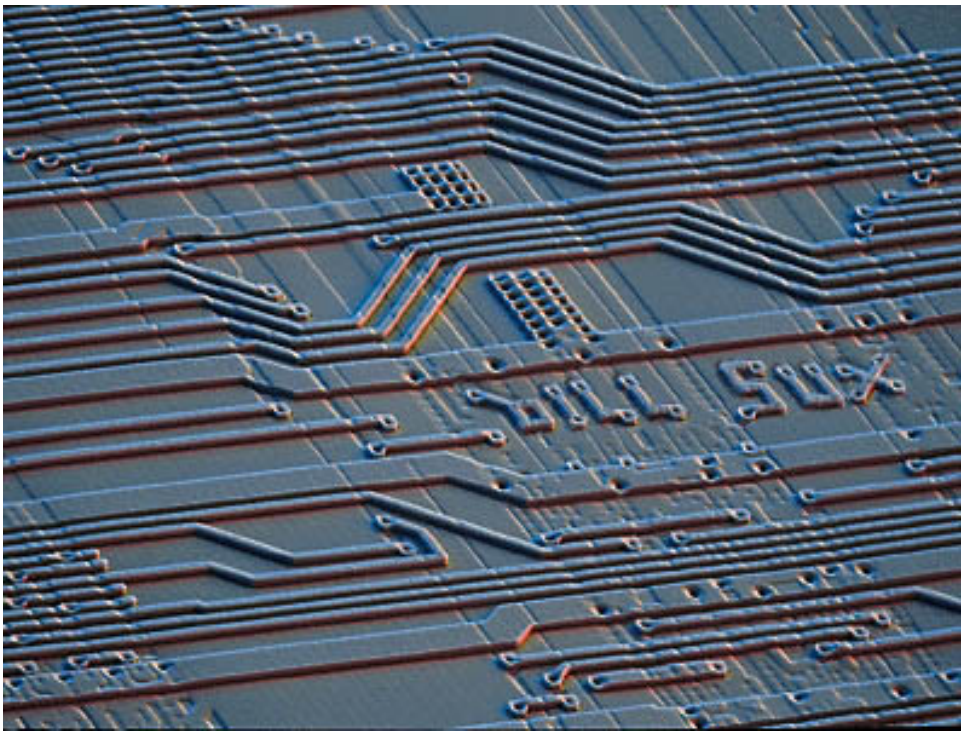
- Undervisningsmateriale til datalogi i gymnasiet

faraz@butt.dk — **Faraz Butt**
mads@danquah.dk — **Mads Danquah**
doktor@dyregod.dk — **Ulf Holm Nielsen**

Vejleder:

keld@ruc.dk — **Keld Helsgaun**

3. januar 2002



Roskilde Universitetscenter
Naturvidenskabelig Basisuddannelse Hus 14.2
3. semester

1 Abstract

Projektets formål har været at udarbejde et undervisningsforløb i grundlæggende maskinarkitektur til gymnasiet. Rapporten gennemgår en række lærings-filosofier heriblandt konstruktivisme, positivisme, empirisme, behaviorisme samt undervisnings-metodik. Der blev udarbejdet et elevmateriale, læremateriale,et forslag til afvikling af undervisningsforløbet, samt program til simulering af en simpel computer. Undervejs argumenteres der for brugen af de forskellige teorier til de forskellige faser af udarbejdelsen.

Nøgleord: Konstruktivisme, Positivisme, Undervisning, Datalogi, Gymnasium.

The goal of this project was to develop a study in basic machine architecture for use in high-school. The paper examines a number of learning-philosophies, including constructivism, positivism, epirism, behaviorism and teaching-methodology. A curriculum and a teacher-guide, a recommendation on how to carry out the lessons was prepared along with a program able to simulate a simple computer. Throughout the paper arguments for using the various theories at various stages are discussed.

Keywords: Constructivism, Positivism, Teaching, Computer Science, High-school

Indhold

2 Forord

Denne rapport giver et indblik i de forskellige undervisnings- og læringsteorier der kan tages i brug i forbindelse med udarbejdelse og formidling af et undervisningsmateriale. I forbindelse med rapporten er der udarbejdet lærer/elev-materiale beregnet til undervisning i grundlæggende computerarkitektur i gymnasiet.

Rapporten henvender sig til primært til undervisere i gymnasiet, men da rapporten beskæftiger sig med undervisning og læring generelt, kan den uden problemer læses af personer der ikke underviser i gymnasiet. De anvendte fremgangsmåder og teorier er ikke bundet til datalogi, og kan derfor med fordel anvendes i andre sammenhænge.

Vi vil gerne takke Thorkild Skjeldborg, for inspiration og rådgivning under udarbejdelse af denne rapport.

3 Indledning

Rapportens mål er at udarbejde et lærer/elev-materiale, samt gøre os overvejelser over hvordan et undervisningsforløb i grundlæggende computerarkitektur kan afvikles. For at være i stand til dette vil der først blive gennemgået en række forskellige læringsteorier, hvorefter der fortsættes med en gennemgang af undervisnings-metodik. Teorien følges op med udarbejdelsen af selve undervisningsmaterialet. Der vil her blive gjort forskellige overvejelser, hvor den netop gennemgåede teori vil blive taget i brug. I slutningen af rapporten vil der blive givet et forslag til hvordan undervisningen kunne tænkes afvikles. Rapportens hovedvægt ligger på udarbejdelsen og fastlæggelse af et undervisningsforløb, og ikke udførelsen af denne.

Rapportens problemformulering lyder:

- *Vi ønsker at udarbejde et undervisningsforløb beregnet på undervisning i grundlæggende computerarkitektur til Gymnasiet. Undervisningens mål skal være at give eleverne en forståelse af grundlæggende computerarkitektur.*
  - *Hvilke overvejelser bør man gøre sig under udarbejdelsen af et sådan forløb*

- *Hvordan kan man afvikle et sådan undervisningsforløb ?*

Følgende emner og problemer behandles i rapporten

- Forskellige holdninger til læring
- Undervisnings-metodik
- Fastlæggelse af mål for undervisning
- Fastlæggelse af teori
- Udarbejdelse af passende opgaver
- Afvikling af et undervisningsforløb i grundlæggende computerarkitektur

4 Læring

Lige siden tidernes morgen har mennesket evnes til at lære nye ting været en afgørende faktor i vores udvikling og overlevelse. Dengang var det evnen til at kunne skaffe føde og forsvare sig der afgjorde hvordan man klarede sig. I dag er det især uddannelse, der bestemmer hvordan et menneske klarer sig i hverdagen.

At læring er en så vigtig del af et menneskes udvikling, har vi været klar over i lang tid. Igennem tiderne har der derfor også været mange forskellige holdninger til hvordan mennesker lærer. I dette afsnit vil vi beskrive en række af de forskellige teorier der igennem tiden har været omkring undervisning og læring.

4.1 Empirisme og Rationalisme

Der findes et væld af forskellige holdninger og teorier der behandler læring, grundlæggende falder de alle mere eller mindre inden for to kategorier[Sjø99]:

- *Empirisme* mener at viden kommer ”udefra”, og opfattes v.ha. vores sanser, handling, og erfaringer. Deraf navnet fra det græske ”empeiria” = erfaring. Viden betragtes som noget objektivt der på forhånd eksistere uden for individet, og skal erkendes igennem observationer og undersøgelser.
- *Rationalisme* mener at viden kommer indefra, at den er noget der bliver til under vores erkendelse af verdenen omkring os. Viden er noget vi selv *konstruere* ud fra vores observationer, og vil derfor til en vis grad være noget subjektivt.

Begge holdninger er enige om at viden er noget der observeres, men de skiller sig når det kommer til hvor denne videns oprindelse skal findes. Om den er noget der i forvejen eksistere og skal opdages, eller noget det enkelte individ ”opfinder”.

4.2 Positivisme

Begrebet positivisme blev først brugt af August Comte (1798 - 1857)[Sjø99]. Positivisten betragter sine omgivelser objektivt og neutralt, grundholdningen i positivisme følger empirismen og er at al viden på forhånd eksisterer. Naturlove og lignende er altså ikke noget vi opfinder, men er en række love der allerede ligger i naturen, vi skal bare finde dem. Ud fra dette synspunkt

arbejder positivisme altid imod at sandsynliggøre ting. Hvis man tilstrækkeligt mange gange kan vise at en sten f.eks. falder til jorden, må man gå ud fra at der eksistere lov der siger at den skal falde. Man kunne fristes til at sige at positivismen betragter alt omkring sig meget naivt og altid positivt (deraf navnet *positivisme*). En positivist vil altid søge at betragte alt objektivt, uden nogle forud-indtagelser eller forventninger.

Ideen er at hvis man kan påvise en ting uden at bygge den på påstande og antagelser, må den være sand. Man kan ikke diskutere hvorvidt en sten falder eller ej. Men hvis man derimod startede med at antage at f.eks. alle grå ting faldt ville man hurtigt kunne diskutere det.

Denne fremlæggelse af positivisme er lidt banal, men man skal ikke glemme at mange af de teorier vi i dag bygger vores viden på, i høj grad er startet som positivistiske eksperimenter. Vi vil senere argumentere både for og imod positivisme.

4.3 Behaviorisme

Positivismen har sit udgangspunkt i naturvidenskab. Men i psykologien er positivisme også blevet benyttet i høj grad. I denne forbindelse opstod begrebet *Behaviorisme*.

I behaviorismen betragter man personer fuldstændigt objektivt. Man prøver beskrive personen udelukkende via ren observerbar adfærd. Når behaviorismen skal beskrive læring, vil den derfor prøve at gøre det ved klart defineret objekter der påvirker hinanden. Således vil læring blive beskrevet som en slags påvirkning (stimulus) der resultere i at eleven lære (respons). Det behavioristiske (og dermed også positivistiske) syn på læring er at da viden er objektivt, og blot er noget der skal erkendes, kan man flytte denne erkendelse fra et individ til et andet. En ofte brugt metafor er at læring er som at hælde væske fra en kande(læreren) ned i en tom krukke(eleven). Det er i forbindelse med denne opfattelse af viden, som en slags stof der kan flyttes, at udtrykket *indlæring* er fremkommet. Man anser viden som en ting der puttes *ind* i hovedet på eleven. Hvis man senere ønsker at undersøge om en elev har lært stoffet, kan man undersøge det på samme måde som man puttede det ind i eleven, nemlig ved at stille eleven et klart spørgsmål (stimulus), og derefter se om eleven svare korrekt (respons). Viden bliver derved noget klart defineret der kan måles ved tests.

4.4 Jean Piagets Konstruktivisme

Jean Piaget var en af de helt store personer inden for 1900-tallets pædagogik, og hans teorier har påvirket pædagogik helt op i vores tid. Allerede som 11-årig skrev han sin første videnskabelige artikel, og som 21 årig fik han en doktorgrad i biologi. Hans afhandling drejede sig om visse vanddyrs reaktioner og tilpasning til de vandløb de levede i. Få år senere begyndte han at interessere sig for det der skulle blive hans livsværk, nemlig hvordan viden opstår og udvikler sig.

Piaget gjorde oprør i mod den dengang meget positivistiske og behavioristiske pædagogik, ved at betragte læringsprocesser på en helt ny måde. Mens Positivismen følger empirismen og siger, at viden er noget objektivt, der befinder sig uden for mennesket, mente Piaget, at viden var noget, man selv konstruerede og blev til i det enkelte individ . Deraf kom navnet på hans teori, ” Konstruktivisme”. I følge Piaget er al viden til en vis grad subjektivt, og den endelige fortolkning af den afhænger af det enkelte individ.

Piaget er mest kendt for er hans stadieteori. I forbindelse med en lang række iq-tests på børn i forskellige aldre lagde Piaget mærke til en sammenhæng imellem de spørgsmål børnene svarede forkert på. Børn på samme alder havde problemer med typer af opgaver. Ud fra disse observationer, udformede han følgende teori.
Et barns udvikling forgår i stadier. Hver stadium kendetegnes ved en bestemt type logik, der er barnets egen form for fornuft:

De 4 stadier er[Sjø99][Egi00]:

- Det sensomotoriske stadie(0-2 år)
- Det pre-operationelle stadie (2-7 år)
- Det konkret-operationelle stadie (7-11 år)
- Det formelt-operationelle stadie (fra 11 år)

På det sensomotoriske og preoperationelle stadie, opfatter barnet alt ud fra at egocentrisk synspunkt. Med egocentrisk menes der at barnet kun kan se ting ud fra sit eget synspunkt, og kan ikke forstå at ting kan ses anderledes. Børn på dette stadie har også en tendens til at opfatte alt animistisk[1], man

[1] Animistisk betyder levende, barnet mener altså alt er i live

kan derfor observere et barn skyde skylden på en sten hvis det gør ondt når det sparker til den. På det konkret-operationelle stadie er banet istand til at klassificere (f.eks. i stand til at ordne klodser efter farve eller form) og serieordne (i stand til at sortere ting). På det formelt-operationelle stadie er barnet i stand til at tænke abstrakt, kontrollere variable osv. For undervisning betyder det at man bliver nød til at undervise efter det stadie eleven befinder sig på. Det hjælper f.eks. ikke at præsentere en person der kun er i stand til at tænke konkret-operationelt for en ligning og bede personen om at forklare hvad den beskriver.

Piaget genovervejede senere sin stadie-teori. Han opdagede efter et stykke tid at man ikke kunne binde stadierne sammen med et menneskets alder. Således er man ikke nødvendigvis i stand til at opfatte alting formelt-operationelt når man er over 11 år, det ville jo svare til at man i al undervisning kunne arbejde med formler og ligninger når blot eleven var over 11. Piaget nåede frem til den konklusion at man kan befinde sig på forskellige stadier alt efter hvilket stof man arbejder med. En fysiker kan således befinde sig på det formelt-operationelle stadie når det kommer til sit fag, men kun på det konkret-operationelle stadie når det kommer til madlavning.

Piaget mente at et hvert menneske prøver at opnår en ligevægt imellem hvad han kaldte *akkomodation* og *assimilation*. Når et menneske tilegner sig viden får det en række indtryk via sine sanser, disse bliver i hjernen behandlet, og sammenholdt med den viden det i forvejen har. Hvis den nye viden stemmer overens med den viden mennesket i forvejen har opbygget, f.eks. at det at en sten falder stemmer overens med viden om tyngdekraft, bliver den assimileret, dvs. indordnet i de tanke-strukture der allerede findes. Hvis den nye viden ikke stemmer overens, vil den blive akkumuleret. Ved akkumulation bliver forgåede tanke-strukture ”revet” ned ,eller ombygget, for at få de nye til at passe ind. Dette er især hvad der sker når man får en elev til at ændre holdning om et emne, f.eks. når eleven lige pludselig forstår hvordan en universal-indikator fungere i kemi.

Ifølge Piaget findes der 4 forskellige typer af assimilation [Egi00]:

- Gentagelse af samme bevægelse således at den fungere mere og mere smidigt og automatisk.
- Genkendelse hvorved det opfattede opleves som noget i forvejen kendt.
- Udforskning af noget man delvist genkender, eller som vækker ens

nysgerrighed med hensyn til om det virkelig kan være som man tror det er, hvorved allerede eksisterende tanke-strukture bliver mere komplicerede.

- Samordning af bevægelses- og tankeskemaer som tidligere ha fungeret uden samordning med hinanden.

Konstruktivismen en i det hele taget meget præget af at Piaget var biolog. Hans teorier om menneskets stræben imod ligevægt imellem akkomodation og assimilation, kan sammenlignes hvordan bløddyr i vandløb, ligeledes søger at opretholde en ligevægt mellem forskellige processer. Og man må antage at det er herfra at Piaget har hentet sin inspiration

4.5 Fordele og ulemper ved de forskellige -ismer

Alle de ovennævnte -ismer har både deres stærke og deres svage sider. Generelt for dem alle er at man ikke kan følge dem rendyrket, men helst skal kombinere de forskellige efter behov, vi vil her prøve at give en oversigt over fordele og ulemper ved de forskellige -ismer.

4.5.1 Positivisme

Positivismen følger rationalismen, der siger at man skal opfatte alting objektivt. Man kan altså ikke tillade sig at lade personlige holdninger eller meninger påvirke hverken eksperimentet eller resultatet. Positivismen bygger på følgende antagelser :

- Man starter med en observation, denne er helt uafhængig af eventuelle teorier.
- Observationen er til hver en tid et sikkert grundlag, og kan ikke diskuteres
- Ved hjælp af induktion kan vi ud fra observationerne udlede generelle regler, når vi blot har foretaget tilpas mange observationer.

Vi vil nu først klarlægge hvad der kan være af problemer ved disse punkter, og derefter argumentere for de nyttige sider ved positivisme og behaviorisme.

Fordelen ved denne holdning er som før nævnt at en ren objektiv observation ikke kan diskuteres, eller modbevises. Men særligt det sidste punkt giver problemer. Der har i lang tid været tradition for at bruge induktion inden for naturvidenskaben til at bevise eller sandsynliggøre sin teori. Men induktion har sine svagheder. For det første kan man med induktion aldrig opnå en endelig sandhed. Lad os f.eks. tage eksemplet fra før med den faldende sten :

Vi slipper en sten 100 gange, hver gang falder den til jorden. Vi prøver med 100 forskellige sten, med 100 forskellige personer, osv. Og hver gang falder stenen til jorden. Ud fra dette kunne man jo slutte at alle ting falder. For at gøre os mere sikre i vores sag kunne vi prøve med grene, jord, og en masse andre ting. Og for hver gang vi afprøvede vores teori ville vi være mere og mere sikre. Men hvis man en dag i stedet for en sten prøvede med en helium-ballon ville den stige til jorden, og alle vores tidligere forsøg ville falde til jorden. Problemet med induktion er netop at man kun kan sandsynliggøre, aldrig bestemme. Man kan afprøve sin teori v.ha. induktion nok så mange gange, men man kan til en hver tid risikere at støde på en ballon der stiger frem for at falde.

Herefter kikker vi på antagelse nr. 1, at vi starter med en observation, og foretager denne uafhængig af teori. Men, hvis vi ikke kan tillade os at antage visse ting, før vi kaster os ud i eksperimentet, vi det give visse komplikationer. Hvad skal man f.eks. vælge at variere i forbindelse med eksperimentet ? Forsøgspersonens tøj?, farven på det man lader falde ? hvilken ugedag forsøget bliver udført ?. Dette er selvfølgelig ekstreme forslag, men det viser at man bliver nød til at gøres sig antagelser og overvejelser, før man begynder eksperimentet.

Og tilsidst problematisere det de to ovenstående argumentationer argument 2. Hvis vi antager at enhver observation er et sikkert grundlag for viden, må vi jo først gå ud fra at denne observation er fuldstændigt objektiv, og entydig. Men, hvis vi på forhånd har antaget en række ting, vil det ikke være muligt at lave en fuldstændig objektiv observation, og da slet ikke at slutte objektiv, og sikker viden ud fra denne. Vi lader stenen falder, men antager samtidigt at dens fald ikke har noget med dens farve at gøre, og vi kan deraf slutte at ens sten med forskellig farve. Men dette sker på grundlag af vores antagelse, og dermed er vores konklusion ikke rent objektiv.

Et af hovedproblemerne ved positivisme er at man sjældent kan bruge en

ren positivistisk konklusion til særligt meget. Konklusionen ”alle sten falder” er jo ganske rigtig, men man kan næppe bruge det til ret meget uden at bygge videre på det. Dermed slet ikke sagt at positivistiske konklusioner ikke kan bruges til noget. Tværtimod kan man med positivismen forsøge at sandsynliggøre ting der ikke på stående fod kan bevises. Man behøver ikke at have en teori om tyngdekraft, eller have lavet teorier om massetiltrækning for at se om alle sten falder.

Positivismen bruges også indirekte i undervisning, nemlig v.ha. behaviorismen. Positivismen bruges direkte når der skal udføres et eksperiment i forbindelse med undervisningen. Det er da også i naturvidenskaben og eksperimenternes verden at positivismen har sit udspring.

4.5.2 Behaviorisme

Behaviorismen kan virke meget primitiv, og næsten ubrugelig, men på trods af dette har troen på at hvis man stimulerede eleverne korrekt, kunne man få dem til at reagere ligeledes korrekt drevet planlægningen af undervisning i USA i mange år[Sjø99]. Men der er en del problemer ved denne form for undervisning. I mange tilfælde vil denne type undervisning resultere i at eleverne i høj grad ledes til at lære udenad. De trænes i at reagere på en bestemt måde, når de ser et bestemt problem. Det kunne f.eks. være at kunne nedskrive $\frac{-B\pm\sqrt{D}}{2A}$ når eleven bliver bedt om at løse en andengradsligning. I dette tilfælde er det jo udmærket, eleven løser problemet korrekt, og kan fortsætte. Men anderledes svært bliver det for eleven hvis han/hun bliver stillet over for et problem og bliver bedt om at løse det, vel og mærke uden at blive fortalt at der er tale om et problem der kan betragtes som en andengradsligning. Lige pludselig får eleven et forkert stimulus, og kan derfor ikke komme med det samme respons.

En af det klare fordele ved at undervise ud fra at behavioristiske synspunkt er at man ikke behøver at tage hensyn til eventuel viden eleverne skulle have i forvejen. Undervisningen vil blive meget lærecentreret, en typisk form for lærecentreret undervisning er forelæsninger.
Behavivioristisk/positivistisk undervisning, med forelæsninger som eksempel, er godt til en række situationer [HCR98] :

- Hvis man ønsker at indføre eleven i et nyt stofområde, og/eller hvis man ønsker at pege på væsentlige problemer i et emneområde, som eleverne ud fra deres erfaringer ikke umiddelbart ville kunne identifi-

cere.

- Hvis den eksisterende litteratur er uigennemskuelig eller utilstrækkelig
- Brugt som introduktion i et nyt stofområde kan forelæsninger virke inspirerende på elevernes selvstudier.
- For at give eleverne omtrent samme grundviden om et emne som forberedelse på en diskussion.
- Når der skal indøves en standard-metode til at løse en bestemt form for opgave. Især når disse skal løses på præcist samme måde, eller hvis de skal løses hurtigt.

På den anden side har de følgende svagheder

- Tillader ikke eleven at være kreativ eller handle på egen hånd.
- Det kan være meget uheldigt at tvinge elever til at lære stof udenad, hvor det er meningen at eleven selv skal være istand til at bygge videre på denne viden senere.

Et godt eksempel på hvor behavioristisk undervisning bliver brugt meget effektivt er træning af piloter. I mange tilfælde er der ikke tid til at piloten tænker over sine handlinger, og derfor trænes piloten til at reagere prompte på visse stimuli.

4.5.3 Konstruktivisme

Konstruktivismen følger empirismen, og mener dermed at viden i sidste ende er subjektivt, og noget der bliver til hos det enkelte individ. Konstruktivismen arbejder ud fra ideen om at eleven selv opbygger sin viden, og dermed selv bliver ansvarlig for egen læring. Dette kan give en række problemer:

- Da eleven selv er ansvarlig for læring, må læreren til en vis grad acceptere at han/hun ikke kan styre eleven frem til målet med undervisningen. Men i stedet må indtage en slags vejledende rolle.
- Da viden er noget der bliver konstrueret hos det enkelte individ kan man risikere at det er den forkerte viden der bliver opbygget.

- Da viden nu er et subjektivt fænomen, kan man i yderste konsekvens ikke tale om korrekt viden, da det er op til det enkelte individ at fortolke denne.

Problemerne kredser alle om at man ikke udefra kan kontrollere et menneskes læring. I sidste ende er det op til det enkelte individ, og hvis en elev f.eks. ikke er motiveret for at lære, kan man ikke lære eleven noget før han/hun er blevet motiveret for det igen. Ligeledes kan man ikke endeligt kontrollere hvilken viden der vil opstå hos den enkelte elev. Dette gør alt i alt at man ikke kan bruge konstruktivismen i praksis i sin reneste form, da man da i princippet må acceptere at alle har kan have hver sin ide om hvordan verden fungere, og at den ene ikke kan være mere rigtig end den anden.

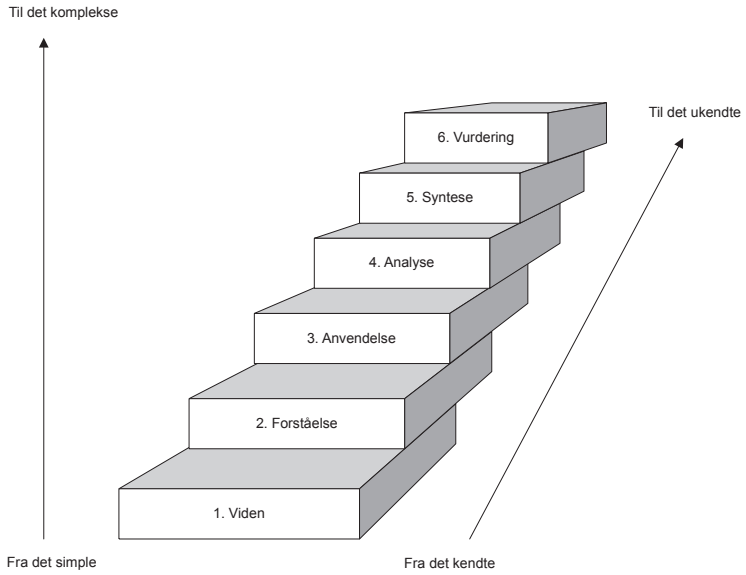
Konstruktivismen har en række tilsvarende fordele, der især kredser om at eleven selv er ansvarlig for sin læring, og selv er med til at opbygge sin viden.

- Eleven drives frem af nysgerrighed. I stedet for at skulle læse noget der er instrueret af en lærer, læser eleven for at kunne bygge videre på sin egen viden
- Ny viden bygger så vidt muligt videre på gammel viden, der ved føler eleven i langt højere grad at der er en rød tråd igennem f.eks. pensum.
- Da eleven selv er med til at bygge sin viden op vil han/hun senere hen i langt højere grad være i stand til at benytte denne viden kreativt.

5 Blooms kognitive taxonomi

Ved den amerikanske psykologforenings årsmøde i 1948 blev der nedsat en arbejdsgruppe hvis formål var at forbedre de eksisterende beskrivelser af undervisningsmål. Dette udsprang i ønsket om at forbedre evaluering af undervisning. Ideen var at hvis man mere nøjagtigt kunne beskrive hvad eleven skulle nå med undervisning ville man bedre være i stand til at måle om eleven havde nået målet. Professor Benjamin Bloom var sekretær for gruppen, og den taxonomi[2] der blev udarbejdet i gruppen mens han var sekrætær[3] er senere hen blev brugt så populær at den har fået navnet Bloom kognitive taxonomi.

Som navnet antyde omhandler Blooms kognitive taxonomi(bkt) et individs læring og anvendelse af viden. Bkt er bygget op som en trappe, hvor hvert trin repræsentere individets udvikling.



Figur 1: Blooms taxonomi

[2] En taxonomi er et klassifikations-system

[3] arbejdsgruppe udarbejdede også en psukomotrisk, og en affektiv taxonomi med henholdsvis Simpson og Krathwohl som sekretær

Trappen består af følgende 6 trin[HCR98]:

1. Begrebet *viden* er et udtryk for et mål, hvor den præstation, der forvente af eleven, er, at han/hun er i stand til at genkende eller gengive det vi kalder faktaviden. det er altså faktaviden, således som den opfattes i vort samfund her og nu.

2. Begrebet *forståelse* udtrykker, at eleven nu har nået et mål, hvor den rene reproduktion er et passeret stadium. Eleven kan altså anvende den indlærte faktaviden i simple, kendte tilfælde. Eleven er sagt med andre ord i stand til at vise sin forståelse af den lærte faktaviden og kan evt. med enkle argumenter gøre rede for brugen af den.

3. Begrebet *anvendelse* er lidt uheldigt valgt da der er tale om anvendelse på alle 6 trin af taxonomien. Med anvendelse menes der på dette trin at eleven er i stand til at anvende sin lærte og forståede viden i simple nye sammenhænge. Dette er elevens første trin til at kunne operere abstrakt med sin nye viden.

4. Begrebet *analyse, syntese og vurdering* dækker trin 4, 5 og 6. På disse trin er eleven nu blevet i stand til at anvende sin faktaviden i mere komplicerede sammenhænge. Analyse betyder at eleven er i stand til at opløse faktaviden i grundbestanddele, syntese betyder at eleven kan sammensætte ny faktaviden af disse grundbestanddele, og vurdering at eleven kan bestemme værdien,og gyldigheden af den syntiserede viden. Disse tre trin er det højst tænkelige mål der kan nås med hensyn til tilegnelse af faktaviden.

Hensigten med på denne måde at opdele stadierne i forståelse af faktaviden, er at gøre det muligt for lære såvel som elev, at afgøre på hvilket trin eleven befinder sig. På denne måde har man et middel til at afgøre om et undervisningsforløb har virket som ønsket, altså om eleven var på det ønskede trin efter forløbet.

6 Metode og metodik

Ordet metodik dækker over teori om hvilke metoder, og hvilke teknikker der kan anvendes for at løse problemer af forskellig karakter.I forbindelse med undervisning metodikken omhandle tilrettelæggelse og gennemførelse af undervisning under forskellige omstændigheder.

Den logiske opbygning udfolder sig i en sammenhængende struktur af arbejdsopgaver, som igen kan henføres til forskellige faser i undervisningen.

Metodikken hjælper til at definere kæde af argumenter i et undervisningsforløb, der fører fra problemformuleringen til konklusion og perspektivering.

6.1 Hvordan skal undervisningen tilrettelægges

Når man som læreren har taget stilling til det udvalgt stof som skal behandles, kan man udfolde følgende spørgsmål [HCR98]:

F = hvordan skal stoffet formidles til eleverne?
A = hvordan skal eleverne aktiveres?
S = hvordan skal undervisnings-forløbet styres?
E = hvordan evaluerer læreren på en passende måde elevernes udbytte af og tilfredshed med undervisningen.

De tre første punkter (F-A-S) omhandler beslutninger der direkte har noget med undervisningen at gøre, både forberedelsesmæssigt, og udførselsmæssigt. Punkt E beskæftiger sig udelukkende med evaluering af undervisningsforløbet. Evalueringsovervejselserne er delt op i to dele, dels må man sørge for at tilrettelægge undervisningsforløbet således at det rent faktisk kan evalueres, og dels må man udforme et evalueringsforløb der kan give et tilfredsstillende resultat.

6.2 Hvordan formidler vi undervisningens indhold? (F)

Når vi snakker om formidling, er det ofte tale om kommunikation via mundtlig overførsel. Men når det gælder et undervisningsforløb sker kommunikationen på flere forskellige måder. Her kan der være tale om formidling via en forelæsning, via en udleveret tekst, en diskussion med en enkelt elev, eller en klasse diskussion. Generelt kan man tale om envejs-, tovejs- eller flervejs-kommunikation.

6.2.1 Forelæsning

Som nævnt i 4.5.2 har forelæsning en række fordele, selvom denne efterhånden har fået et ry af at være forældet og ubrugeligt. Grunden til dette er at forelæsninger let kan bruges forkert, hvis man ikke har en af de førnævnte begrundelser, bør man kraftigt overveje om der ikke er et alternativ. Når forelæsninger benyttes bør man holde sig til en række praktiske retningslinjer: [HCR98]

- Vælg et passende tempo og indlæg korte pauser med jævne mellemrum.
- Man skal udtrykke sig klart og tydeligt med en passene stemmestyrke.
- Man må bestræbe sig på at der ikke er faktorer der kan distrahere eleverne, herunder personlig fremtoning, og lokalevilkår.
- Man må så vidt muligt bestræbe sig for at holde øjenkontakt med eleverne. Selvom der er tale om envejs-kommunikation kan man stadig få et vist feedback ved at iagttage eleverne.

  Man kan f.eks. udvælge 4 elever spredt rundt i rummet, og derefter lade blikket vandre imellem disse 4. De resterende elever vil dermed få en fornemmelse af at man søger øjenkontakt med dem.
- Man kan med fordel benytte sig af audiovisuelle hjælpemidler.
- Det er vigtigt at stoffet, og ikke læreren holdes i centrum. Hvis stoffet formidlers klart og sammenhængende, med vægt på hvor de vigtige dele af stoffet er. Det er ikke læreren job at underholde eleverne for at ”snyde”dem til at lære, men derimod stoffet selv der skal fremlægges på en interessant måde således at eleverne selv føler sig underholdt.
- Forelæsningen bør startes med en kort introduktion til hovedlinjer og væsentlige problemer i dagens stof. På den måde får eleverne mulighed for at overskue stoffet, og bedre mulighed for at forholde sig til stoffet undervejs.
- Forelæsningerne skal planlægges med hensyn til elevens forudsætninger.

Det er altafgørende at fokuset ligger rigtigt, og at elevernes interesse fastholdes, da man ved envejs-kommunikation i langt højere grad risikere at ”miste” elever undervejs [HCR98]

6.3 Gruppearbejde

Forskellige opgaver kræver forskellige former for gruppearbejde. Man kan opdele gruppearbejde i produktivt arbejde, og reproduktivt arbejde. I reproduktivt-arbejde skal gruppen afprøve en teori, gruppens skal reproducere noget i forvejen afprøvet. I produktivt arbejde skal gruppen selv fremkomme med en løsning på et problem. Det er vigtigt at erkende at langt fra alle opgaver egner sig til gruppearbejde. Generelt kan man sige at opgaver der går ud på at genkende og gengive[4] hovedsageligt egner sig til enkelemandsarbejde. Hvis opgaven sigter imod at eleven skal opnå forståelse og evt. kunne anvende[5] vil den egne sig bedre til gruppearbejde. Som med så mange andre regler inden for undervisning bør man dog altid vurdere hvad der er egnet til hver enkelt situation frem for blindt at stole på "teorien".

6.3.1 Reproduktivt gruppearbejde

Denne form for gruppearbejde egner sig bedst til små gruppe på 2-3 mennesker. Det er gruppens opgave at gentage et forsøg/en metode som læreren i forvejen har dokumenteret og/eller demonstreret. Argumentet for at gruppen ikke bør overstige 3, er at der ikke arbejdes med et materiale der for alvor kan diskuteres. Dog kan det være en fordel at diskutere korrektheden af forsøgs-data med en eller to personer. Målet med reproduktivt gruppearbejde er at elverne får mulighed for at gennemprøve, og forstå teori. Det er derfor også vigtigt at opgaven er stillet på en sådan måde at dette sker. Dette gøres f.eks. ved at formulere problemet således at eleven kan se meningen dette.

6.3.2 Produktivt gruppearbejde

Produktivt gruppearbejde egner sig til grupper på 4-6 elever. I produktivt gruppearbejde skal gruppe selv arbejde sig frem til en løsning på et problem. Svaret, hvis der da findes et sådant, foreligger ikke fra starten, men skal udarbejdes v.ha. samarbejde i gruppen. De enkelte gruppemedlemmer har her mulighed for at trække på de andre medlemmer, og derved dække et større område end man ville kunne med enkeltmands-arbejde. Denne type opgaver egner sig klart bedre til gruppearbejde en den reproduktive, da man her kan være ude for at problemet slet ikke kan løses af een mand alene på den angivne tid. Igen er det vigtigt at opgave stilles korrekt, den

[4] Niveau 1 Bloom's taxonomi, se side 5
[5] niveau 2 og 3 i Bloom's taxonomi

bør være meningsfyldt for eleverne, og skal opfordre til diskussion og forskellige standpunkter. Dette kan opnås ved f.eks. at stille en opgave der provokere eleverne ved at stride imod dennes opfattelse.

Formålet med de produktive gruppeopgaver er dels at få eleverne til at få indsigt i de forskellige problemstillinger omkring et stof. Men formålet er i høj grad også at give eleven forståelse for at han/hun selv kan danne viden v.ha. diskussion og reflektering over et stof.

6.4 Elevaktivering (A)

Det at man får eleverne til at lytte, tage notater, læse, skrive på tastaturet, kaldes for fysiske aktiviteter. Psykiske aktiviteter indebære at tænke, indøve og huske. Det vil sige at i enhver undervisningssituation foregår der både fysiske og psykiske elevaktiviteter. Man kan så spørge sig selv, i hvilke situationer lærer eleverne bedst i? Og til det har forskere udarbejdede fire rådgivningspunkter: Udgangspunktet af de fire råd er taget i elevernes erfaringer. Omgivelserne og undervisningsprocessen er med til at give påvirkninger af de erfaringer eleverne får.
De fire råd[HCR98]

1. Eleverne må have lejlighed til at praktisere den form for adfærd, man sigter imod som mål for undervisningen.

2. Elevernes tilegnelsesoplevelse skal være sådan, at de finder tilfredsstillende ved den adfærd, som er målet. Interesse og motivation spiller her en hovedrolle.

3. De mål, man ønsker at opnå, skal ligge inden for elevernes muligheder. Det nytter altså ikke noget at lade eleven stræbe imod et mål eleven ikke kan nå.

4. Understregning af det hensigtsmæssige i at variere undervisnings og arbejdsformer.

5. Råd til læreren, er en understregning af, at oplevelsessituation som regel har flere indlærings resultater. Medens eleverne arbejder i undervisningssituation, lærer de noget om de problemer, de beskæftiger sig med.

Generelt siger konstruktivismen at eleven selv står for at opbygge sin viden,

så hvis eleven ikke er motiveret, eller kan se en mening med det hele vil han/hun ikke få noget ud af undervisningen. Denne motivation kan opnås på en række forskellige måder, vi kommer her med en række forslag, der også er taget i brug i vores undervisningsmateriale :

6.5 Korte forelæsninger

Forlæsninger[6] kan være et godt værktøj til at formidle en mængde teori, men man risikere også at passivere eleverne, og i værste fald at de falder i søvn. For at undgå dette er det vigtigt at man ikke forlæser mere end ca 15 minutter af gangen. Derefter kan man afbryde forelæsningen ved f.eks. at inddrage hele klassen i et opgave, bede dem om at begrunde en påstand man er kommet med, eller lave en leg.

6.6 Lege

En måde at ”snyde” elever til motivation, kan være at gribe stoffet alternativt an v.ha. en leg. I denne rapports tilfælde, kan man konstruere en leg hvor hver elev skal foregive at være et komponent i den simple datamaskine. Eleverne skal nu afvikle et simpelt program, og hver elev må kun udføre et par forskellige ting. En elev kan f.eks. være bussen, og må derfor kun modtage data fra et komponent, og give det videre til et andet. En anden eleven kan være kontrolenheden der fortæller de andre hvad de skal gøre, osv.

6.7 Elev til elev undervisning

Da man utvivlsomt vil komme ud for at elever har forskellige niveauer inden for datalogi, kan man med fordel udnytte at elever kan have lettere ved at forstå stoffet, hvis det bliver forklaret af en jævnaldrende. For at gøre dette starter man med at sortere eleverne efter niveau, dette kan enten gøres manuelt af læreren, eller eventuelt endnu en leg, herefter gives opgaven, og eleverne kan nu hjælpe hinanden frem til løsningen. Den dygtige eleven får trænet sin evne til at formidle sit stof, og den dårligere elev får ekstra hjælp til at forstå stoffet.

[6] Med forelæsninger kan der også forstås envejs undervisning. Hovedpointen er at eleven er passiv.

6.8 Hvordan skal undervisningen styres? (S)

Aktiviteterne i undervisningen bør være hensigtsmæssige både i forhold til det mål, undervisningen stiler imod, og i forhold til rammerne for undervisningen. Man bør tage stilling til følgende punkter:

1. Indhold i time (udvælgelse af stof der passer til eleverne)
2. Undervisningsformen (udvalg af undervisningsform der passer bedst til den givne situation)
3. Valg af passende organisationsform. Skal eleverne arbejde i grupper? individuelt? projektorienteret?
4. Evaluering og kontrol, hvordan evalueres forløbet?
5. Reaktioner på afvigelse fra planen, hvordan skal eventuelle problemer håndteres?

6.9 Evaluering af undervisningen (E)

Det at evaluere noget betyder med andre ord at man systematisk arbejder med at indsamle informationer, at analysere disse og udarbejde en samlet vurdering. Dette kan indeholde elevernes udbytte af undervisningen. Da læring omhandler en række processer vi i sidste ende kun kan gætte på hvordan fungere, er det også vigtigt at erkende, at man i dag ikke kan afgøre med sikkerhed hvorvidt f.eks. en elev har lært et givent stof. Vi kan kun foretage en række tests af dette, og gå ud fra at disse er korrekte. Vi vil ikke beskæftige os mere med evaluering, da vi ikke kommer videre ind på det i vores materiale. Blot skal det endnu engang understreges, at det under enhver udarbejdelse af undervisningsmateriale er vigtigt at definere et klart og ”testbart” mål. Ellers vil enhver evaluering blive gjort totalt umulig !

7 Undervisningsmaterialer

Vi vil nu se på udarbejdelsen af undervisningsmaterialet.

7.1 Overvejelser

Det bedste udgangspunkt man kan tage når der skal udarbejdes undervisningsmateriale, der i sidste ende skal læses af eleven, er elevens. Det er vigtigt at materialet tager udgangspunkt i elevens niveau, og derfra hjælper eleven med at komme videre. Udarbejdelsesprocessen skal sagt på en anden måde være elevorienteret også selvom det resulterende materiale skulle være f.eks. lærerorienteret. Stoffet skal skrives med henblik på at hjælpe eleven frem til en forståelse, og ikke at belære eleven om denne.

Ifølge konstruktivismen, vil en elev bygge al ny viden på gammel viden. Det er derfor vigtigt at undervisningsmaterialet tager udgangspunkt i ting eleven ved, og bygger videre derfra[Lau93]. Det er vigtigt at grundholdningen i undervisningsmaterialet er at eleven skal hjælpes frem og ikke belæres. Ting skal derfor så vidt muligt argumenteres for, frem for blot at blive konstateret. F.eks. ”En CPU fungere på den og den måde, da den skal kunne udføre de og de operationer”, frem for ” En CPU fungere på den og den måde fordi man har valgt det”.

Et andet vigtigt punkt i undervisning generelt er at der skal være et klart defineret mål[Lau93]. En elev kan acceptere at blive undervist i noget de ikke umiddelbart kan forstå, så længe der bare er et klart defineret mål, og dermed mening i undervisningen. Argumentationen ”I skal lære det fordi jeg siger I får brug for det” er altså ikke umiddelbart en bedste. Man vil i langt højere grad kunne motivere eleverne hvis man siger ”Læs kapitlet om computerarkitekture til næste gang, fordi i får brug for det for at kunne lave opgaverne” frem for simpelthen at sige ”læs side 10-20 til næste gang”. Uden et mål er der heller ikke en klar afgrænsning af stoffet, og det kan i sidste ende blive lige så svært for læreren at undervise, som for eleven at forstå stoffet. Den konstruktivistiske argumentation for at der skal være et klart defineret mål er at viden ikke nødvendigvis kan indordnes i elevens tanke-strukture hvis eleven ikke ved hvor denne viden høre til, eller hvad den i sidste ende bør hænge sammen med.

For nu at vende os imod en mere konkret udarbejdelse af undervisningsmaterialet vil vi tage udgangspunkt i den positivistiske undervisningsfilosofi

behaviorisme. Som det før er nævnt vil man i positivisme starte med at se bort fra elevens stadie. Vi starter altså udelukkende med at koncentrere os om hvad der skal læres og ikke hvordan.

7.2 Mål

For at lette udarbejdelsen af undervisningen bør man først starte med at sætte sig et hovedmål med undervisningen. For at nå dette hovedmål skal eleverne opnå en række undermål[Lau93]. Disse undermål skal være[Lau93]

- Så præcist defineret at men let kan afgøre om en elev er i stand til at nå dem eller ej.
- Nødvendige, forstået på den måde at uden det enkelte undermål kan hovedmålet ikke siges at have været nået
- Fuldstændigt, således at de enkelt undermål skal tilsammen dække det vidensgrundlag der implicit er defineret i hovedmålet.

Ved at sørge for at under og hoved-målet tilfredsstiller disse krav, vil man ende med et meget klarere billede af hvad man vil formidle. En fremgangsmåde til at få udspecificerede disse mål er følgende algoritme [Lau93]:

1. Fastlæg målet x. item Definer adfærden, y, der ville vise at man havde nået målet.
2. Er y defineret præcist nok til at en evt. kollega ville være enig i at adfærd y viste at man var nået målet x? Hvis det ikke ville være tilfældet returner da til 2 og redefiner definitionen.
3. Kan målet nås uden at eleven nødvendigvis vil udvise y ? hvis det er tilfælde og y er en brugbar adfærd definer da endnu et mål der passer til y. Retur ellers til 2 og redefiner eller erstat definitionen.
4. Dækker de opsamlede y'er alting implicit i x ? Hvis ikke returner til 2 og lav flere y'er
5. Opstil en liste over mål og opgaver.

7.3 Udarbejdelse

Vi har indtil nu arbejdet behavioristisk, og udeladt al spekulation om hvor eleven befinder sig videnmæssigt når vi påbegynder undervisningsforløbet.

En af nøglerne til effektiv undervisning er at tage udgangspunkt i elevens viden, og bygge videre derfra, vi vil derfor være nød til (så vidt muligt) at sætte os i elevens sted[**?**, metodik]or at kunne udarbejde et undervisnings materiale der tager højde for dette. At sætte sig i elevens sted kan være en meget hård opgave, især for undervisere der til dagligt arbejder med deres stof på et højt niveau. For at forstå hvordan eleverne har det med det for dem, nye materiale, må man prøve at huske tilbage på hvordan man selv forholdte sig til materialet da det blev præsenteret for en. Men må gøre forsøg på at forudse de mest almindelige fejlopfattelser eleverne kunne få af stoffet, og derefter sørge for at udforme undervisningsmaterialet på en sådan måde at man undgår, eller modvirker disse fejlopfattelser.

Man kan forvente sig at eleverne kan deles ind i følgende gruppe:

- De blanke: eleven har ingen tidligere erfaringer med computer og har ikke nogen datalogisk viden man kan bygge videre på.
- De bevidste: eleven har et alment kendskab til computere, de har ikke nogen programmeringskendskaber, men har på den anden side ikke nogen problemer med at arbejde med computere.
- De erfarne: eleven har kendskab til programmering, og kan lige fremmest på forhånd være fuldt ud kendt med stoffet.

Man bør klart rettet ens undervisningsmateriale imod midten af målgruppen, da midten i de fleste tilfælde, og da også dette, vil være den største. Vi går altså ud fra at eleven har følgende egenskaber:

- Har ingen problemer med at arbejde med en computer.
- Har før arbejdet en smule med computere.
- Er interesseret i at lære mere om computere.

Man kan forvente at følgende fejlopfattelser og holdninger vil kunne optræde hos eleven.

- Programmer og data er to separate ting.
- Computeren kan forstå abstrakte kommandoer så som division, eller udskrivning.
- En ”magisk” opfattelse af computeren, forstået på måde at eleven ingen ide har om hvordan computeren fungere internt.

- forstår på den ene side at en computer kan lave beregninger, udføre komplekse operationer osv. Men har på den anden side intet grundlag for at kunne udtale sig om en computers begrænsninger.

Det er sagt på en anden måde vigtigt at man lægger ud med at forklare eleven hvordan computeren fungere på de aller laveste niveauer. For at tage udgangspunkt i elevens niveau må man først gøre sig klart hvad eleven kan forventes at have af forhåndskundskaber. Vi går ud fra følgende om eleverne:

- Kan programmere simple programmer.
- Kender ikke til maskinarkitekture på forhånd
- Kan addere, subtrahere, multiplicere og dividere
- En hvis procentdel af eleverne må kunne antages at være interesseret

Man kan nu v.ha. den før beskrevne algoritme, udarbejde et mål med undervisningen og en række adfærde eleven må forventes at udvise hvis undervisningen lykkes.

1. *Opstil mål:*
   Opnå en forståelse af hvordan computeren fungere på de laveste niveau.
2. *Definer adfærd:*
   Eleven skal kunne opbygge et simpelt program i maskinkode.
3. *Kan eleven opnå målet uden at udvise adfærden?*
   Ja, eleven har ikke nødvendigvis forstået hvordan maskinen fungere på laveste niveau.
4. *Dækker adfærden hele emnet:*
   Nej, f.eks. er dualitet imellem program og data ikke dækket.

Efter at have gennemgået algoritmen et del gange kom gruppen frem til følgende undermål, og tilsvarende adfærd eleven vil udvise når han/hun når dette.

- *Mål:*
  - Opnå en forståelse af hvordan computeren fungere på det laveste niveau.
- *Adfærd:*
  - Kan opbygge en simpel instruktion i mikrokode
  - Er i stand til at overskue og forklare et maskinkode program på papir.
  - Forstå dualiteten imellem program og data

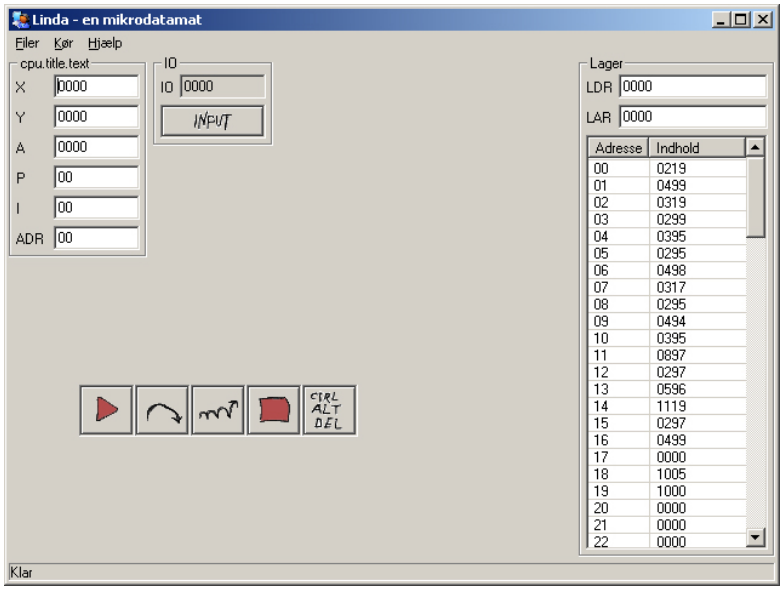
Vi har nu fået defineret vores mål, samt hvilken adfærd man kan forvente eleven vil udvise når denne har nået målet. Vi kan nu gøres os overvejelser over hvilken teori, og hvilke opgaver der skal til få at få eleven til at nå disse mål.

7.4 LINDA - en mikrodatamat

En vigtig del af undervisningen er at eleverne skal være i stand til at afprøve sine program. Gruppen har derfor valgt at udvikle et program der kan simulerer en simpel datamaskine. Grunden til at gruppen valgte at implementere sit eget program, er at der ikke i forvejen findes et program der giver brugeren mulighed for at ændre i implementeringen af instruktionerne. Det gør gruppens program LINDA da kildekoden er frit tilgængelig og koden er lavet for at nybegyndere skal kunne forstå den. Desuden spænder de fleste andre programmer på markedet meget bredt, og megen af den funktionalitet var der slet ikke brug for i dette forløb. LINDA implementerer dog den samme maskinmodel som de andre programmer, nemlig SL69. LINDA er skrevet i Java, kildekoden kan ses i bilag B og selve programmet findes i færdig form på den vedlagte CD (bilag E).

LINDA kan:

- Afvikle et program skrevet i SL69 maskinkode
- Udføre en maskininstruktion af gangen
- Udføre en mikroinstruktion af gangen
- Hente og gemme maskinprogrammer



Figur 2: LINDA

7.5 Teori

Vi tager udgangspunkt i hovedmålet: ”Opnå en forståelse af hvordan computeren fungere på det laveste niveau”.

Teorien er valgt således at den dækker de ønskede mål. Teori, eksempler samt opgaver skal tilsammen give eleven er forståelse af stoffet, teorien er selve stoffet, mens opgaver og eksempler er værktøjer der kan hjælpe eleven til en forståelse.

Da eleven ikke kan forventes at have nogen viden om maskinarkitekture på forhånd, må man sørge for at undervisningsmaterialet starter helt fra bunden, og derefter arbejder sig langsomt opad. Et forslag til dette kunne f.eks. være:

- Introduktion til en simpel computer
  - komponenter
- Programmering af en simpel computer
  - Maskinniveauer
  - Maskinkode

  - Maskinkode i LINDA

- Mikrokode
  - Introduktion til maskinkode
  - Mikrokode
  - Mikrokode i LINDA

På denne måde vil eleven langsomt blive hjulpet frem til en forståelse af maskin- og mikrokode.

7.6 Opgaver

Til teorien bør der følge en række opgaver og eksempler. En række mindre eksempler i løbet af stoffet hjælper eleven til at være sikker på at han/hun har forstået stoffet korrekt. Og en række opgaver der løses i timen i samarbejde med en eller flere, giver elev og lærer et klart mål at gå efter. Opgaverne bør derfor defineres således at de matcher de før fundne adfære/undermål. Vi giver her et par eksempler på sådanne opgaver.

7.6.1 Opbyg en simpel instruktion i maskinkode

Eleven får her stillet til opgave at løse et simpel problem v.ha. en instruktion eleven selv skal konstruere. Det omliggende program er allerede skrevet, elevens opgave er at få det til at virke som ønsket. Opgavens mål er at teste om eleven har opnået en tilstrækkelig forståelse af mikrokode til at løse et problem med dette.

7.6.2 Oversku og forklar et maskinkode program

Eleven får til opgave at forklare hvilke lageradresser et givet program benytter i løbet af sin eksekvering, og hvilke af disse der bliver ændret til hvad. Opgavens mål er at teste om eleven er i stand til at overskue maskinkoden. Det må forventes at eleven kun vil være istand til at give en rigtig besvarelse hvis eleven forstår programmet.

7.6.3 Forståelse af dualiteten imellem program og data

Eleven får til opgave at skulle afgøre hvile dele af lageret der indeholder data som et stillet program benytter sig af, og hvilke dele der ret faktisk er programkode. Programmet er nu konstrueret således at det hopper imellem forskellige adresser og undervejs modificere et stykke ”data” således at det tilsidst indgår som et stykke programkode. Denne opgave vil kræve at eleven forstår at data og programkode i virkeligheden er det samme.

8 Elevmaterialet

På baggrund af det foregående er der udarbejdet et elevmateriale der kan findes i bilag D.

9 Lærevejledning

Det må forventes at eleven særligt i starten af undervisningsforløbet vil befinde sig på det pre/konkret-operationelle stadie(Piaget), eller 2-3 niveau(Bloom). Derfor er det vigtigt at lade eleverne anvende deres nye viden. Timerne skal derfor bære mere præg af at være eksperimenterende, end belærende. Den eksperimenterende stemning kan f.eks. opnås ved at blande alternative elementer så som lege ind i undervisningen.

Tanken er at eleverne i høj grad selv skal være med til at drive undervisningen frem. Da det hele handler om at eleverne skal forstå en række grundlæggende ting, er det vigtigt at de får mulighed og tid for at forstå dette. Det er derfor der skal lægges vægt på det eksperimentelle.

Vi henviser generelt til det udarbejdede læremateriale i bilag C.

10 Diskussion

Vi vil nu tage rapportens problemformulering op til diskussion. Rapportens mål er at få udarbejdet et undervisningsforløb i datalogi, der skal kunne give elverne en forståelse af grundlæggende computerarkitektur. I de foregående afsnit har vi gennemgået en udarbejdelse af undervisningsforløbet, men der mangler stadig en afgørende del, nemlig at afgøre hvor vidt man kan sige at eleverne har opnået en forståelse af stoffet.

Da vi desværre ikke har fået afprøvet forløbet på en gymnasieklasse, er det svært at give et endegyldigt svar på dette.

Undervejs i forløbet skal eleven løse forskellige opgaver. Disse opgaver vil indikere om eleven har opnået en forståelse af de enkelte dele af stoffet. Hvis undervisningsmaterialet er udformet korrekt, og undervisningen forgår på en hensigtsmæssig måde, må vi forvente eleven er i stand til at løse opgaverne på egen hånd. Skulle eleven være ude af stand til at løse opgaverne må fejlen kunne finde et af følgende steder.

- *Materialet*: Hvis den viden eleven skal bruge for at løse opgaven simpelthen ikke findes i materialet, eller hvis undervisningsmaterialet er udformet så uheldigt at denne viden kan misforstås, vil eleven ikke være i stand til at løse opgaver. Dette vil være meget problematisk da materialet netop skal være det sted hvor eleven kan hente sin information fra.

- *Undervisningen*: Hvis undervisningen ikke forgår på en hensigtsmæssige måde, risikere man at eleven får sværere ved at opnås den ønskede forståelse af materialet. Man kan selvfølgelig sige at eleven i princippet selv bør være istand til at opnå forståelsen blot materialet er udlevet til eleven, men som det før er nævnt kan det tit være en fordel at løse problemer i fællesskab med en anden. Skulle der være uklarheder eller misforståelser i undervisningsmaterialet er det også vigtigt at eleven kan støtte sig op af undervisningen.

- *Eleven*: Som før nævnt er elevens motivation en altafgørende faktor for elevens læring. Hvis eleven ikke er interesseret, eller er uengageret vil undervisningen højst sandsynlig mislykkes. Manglende motivation kan være et resultat af de to ovennævnte årsager.

Målgruppen er som tidligere nævnt gymnasieelever. Det må derfor forventes at det udarbejdede materiale dækker den pågældende del af gymnasiebekendtgørelsen. Ifølge bekendtgørelsen[fgu99] skal undervisningen i maskinarkitektur dække følgende:

- Virtuelle maskinniveauer og systemprogrammel
- Dualitet mellem program og data.

Derudover er der også en række forventninger til undervisningsmål, ifølge bekendtgørelsen skal eleven efter forløbet

1. have en datalogisk indsigt, som kombinerer praktiske færdigheder, teoretisk forståelse og generel viden
2. kunne anvende et datalogisk begrebsapparat til beskrivelse og forståelse af informationsbehandling
3. kunne formulere sig skriftligt og mundtligt om datalogiske emner
4. kunne anvende deres kreativitet og abstraktionsevne i arbejdet med datalogiske problemstillinger
5. kende til og kunne anvende datalogiske metoder i problemløsning
6. kende til strukturering, formalisering og modellering og kunne anvende det på konkrete problemstillinger
7. kunne implementere de udviklede modeller ved hjælp af værktøjsprogrammel og generelt programmeringssprog
8. have kendskab til fundamentale algoritmer og datastrukturer
9. have kendskab til eksempler på principielle begrænsninger af computerens anvendelser
10. have forståelse for computerens fundamentale strukturer og processer.

Da det udformede undervisningsmateriale kun beskæftiger sig med et afgrænset område er ikke alle disse mål nået, men dog må de fleste af disse krav siges at være opfyldte. Kravene til teorien må ligeledes siges at være opfyldt.

Det ville have været en stor fordel hvis undervisningsmaterialet kunne have været ”afprøvet” på en gymnasieklasse inden projektets afslutning, men p.ga. tidsnød var dette desværre ikke en mulighed. Man kan snakke nok så meget om hvordan et undervisningsforløb bør afvikles, og hvordan materialet bør påvirke eleverne, men i sidste ende er den eneste rigtige måde at udvikle undervisning, at afprøve det, og derefter tilpasse det virkeligheden. Det må derfor siges at være en brist ved dette projekt, at materialet ikke endnu har været afprøvet.

11 Konklusion

Ud fra den ovenstående diskussion er vi kommet frem til at vi ikke kan give et endeligt svar på om undervisningsmaterialet er tilstrækkeligt eller ej. Hovedproblemet er at forløbet ikke er afprøvet i praksis. Dog mener vi at have opnået en række mål der alligevel sætter os i stand til at komme med en konklusion.

- Vi har udarbejdet et undervisningsforløb med et klart defineret mål.
- Dette mål er opdelt i en række undermål, der alle kan opnås af eleven.
- Teorien påkrævet af hvert af disse undermål er indeholdt i elev-materialet.
- Hvert af disse undermål testes af en række opgaver, således at lære og elev kan sikre sig om disse er nået.
- Ud fra egne vurderinger opfylder undervisningsmaterialet gymnasiebekendtgørelsen.

Vi mener at eleven har nået målet hvis eleven er istand til på egen hånd at løse de stillede opgaver. Vi mener ligeledes at disse opgaver repræsentere en række undermål, og at disse kan nås af eleven v.ha. teorien i undervisningsmaterialet. Vi mener at teorien er gennemgået på en så hensigtsmæssig måde som det har været os muligt. Så alt i alt vil vi vove at konkludere at vi, så vidt det er muligt uden en evaluering af undervisningsforløbet i praksis, har opfyldt vores problemformulering.

12 Perspektivering

Det har besværliggjordt vores evaluering af undervisningsmaterialet at det ikke har været afprøvet i praksis. Det vil derfor være stærkt tilrådeligt at man gør dette hvis man har tænkt sig at lave et lignende projekt, eller evt. bygge videre på denne rapport. Til projektet blev programmet LINDA udviklet. Dette program viste sig at være mere tidkrævende end først antaget, så for at sikre at programmet blev færdigt til tiden er den grafiske brugergrænseflade blevet nedprioriteret, hvilket da også fremgår af udseendet. Dog er programmet fuldt funktionelt, og kan simulerer en simpel datamaskine. LINDA kan findes på den vedlagte CD-ROM i bilag E. Gruppen blev inspireret til at lave dette program af et lignende program, kaldet ”WinDavid” udviklet af John Q. Christensen[7]. WinDavid bruges i skrivende stund i undervisningen i datalogi i gymnasiet, og kan med fordel bruges i udarbejdelsen af undervisningsforløb.

[7] Et orienteringseksemplar kan hentes på http://home.worldonline.dk/ qvottrup/data/

13 Referencer

[Egi00] Henry Egidius. *Pædagogik i det 21. århundrede*. Gyldendalske Uddannelse, 1 edition, 2000.

[fgu99] Område for gymnasiale uddannelser. *Gymnasiebekendtgørelsen - Datalogi*. Undervisnings Ministeriet, 1999. Vedlagt i bilag A, Kan også findes på Internettet:http://www.uvm.dk/gymnasie/almen/lov/bilag5.htm.

[HCR98] Erik B. Yde Hans Chr. Ralking, Thomas Tylén. *Profession lære - 1 metodik*. Erhvervskolernes forlag, 4 edition, 1998.

[Lau93] Diana Laurillard. *Rethinking university teaching*. Routledge, 1993.

[Sjø99] Svein Sjøberg. *Naturfag som allmenndannelse*. Ad Notam Gyldendal, 1 edition, 1999.

14 Supplerende Litteratur

[1] Udvalgt af Albert Chr. Paulsen. Natuvidenskabernes uddannelsesteori, supplerende tekster, 2001.

[2] James L. Antonakos. *An Introduction to the Intel Family of Microprocessors*. Prentice Hall, 2 edition, 1996.

[3] Jøregen E. G. Nielsen Arne Tolstrup Madsen. *DAVID, Datamaskinens fundamentale virkemåde*. Forlaget Systime, 1991.

[4] Barry B. Brey. *The Intel Microprocessors: 8086/8088, 80186/80188, 80286/80386, 80486, Pentium and Pentium Pro processor*. Prentice Hall, 4 edition, 1997.

[5] Michael Cole Denis Newman, Peg Griffin. *The construction zone*. The Press Syndicate of the University of Cambridge, 1 edition, 1989.

[6] Walter Frøyen. *Ansvar for andres læring*. Gyldendal Uddannelse, 1 edition, 1998.

[7] Hans G Furth. *Piaget for teachers*. Prentice Hall, 1970.

[8] Peter B Pufall George Forman. *Constructivism in the Computer Age*. Lawrence Erlbaum Associates, Publishers, 1988.

[9] Knud Illeris. Piaget og pædagogikken, udarbejdelse af en pædagogisk fortåelsesramme i forlængelse af piagets udviklingsteori og erfaringer fra at voksenuddannelsesprojekt, 1995.

[10] Ebbe Vestergaard Jens Kyrstein. *Undervisninglære: En elementær indføring*. Munksgaard, 1978.

[11] Trond Ålvik Kirsten Reisby. *Undervisningslære II*. Gyldendals pædagogiske Bibliotek, 1971.

[12] Trond Ålvik. *Undervisningslære I*. Gyldedals Pædagogiske Bibliotek, 1972.

[13] Berit Rognhaug. *Kunnskap, Teknologi og Læring*. Tano, 1995.

[14] Mihaela Sabin. *Computer Science Education: Literature Survey*. Computer Science Department, University of New Hampshire, 1998.

[15] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Reading, Massachusetts, U.S.A., 3 edition, 1992.

[16] Thorkild Skjelborg. *Datalære - sådan!* Lademann Læremidler, 1987.

[17] Thorkild Skjelborg. *Problemløsning og formalisering med Visual Basic*. Forlaget Sif, 1 edition, 1996.

[18] Andreas Striib. *Undervisning*. Forlaget Klim, 1997.

[19] Barry Wilkinson. *Computer Architecture: design and performance*. Prentice Hall, 2 edition, 1991.

[20] Mehdi R. Zargham. *Computer architecture : Single and parallel systems*. Prentice-Hall, Inc., 1996.

A Gymnasie bekendtgørelse for datalogi

Bilag 5

Gymnasiebekendtgørelsen

DATALOGI

MAJ 1999

1. Identitet og formål

1.1 I datalogi beskæftiger man sig med begreber og metoder, som er fundamentale for forståelse af og deltagelse i den informationsteknologiske udvikling.

1.2 I datalogi beskæftiger man sig med de generelle og universelle principper, metoder og teknikker, som er fælles for hele informationsteknologien og dens mangeartede anvendelser. Det overordnede teoretiske indhold omfatter begreberne information, struktur, proces og model.

1.3 Fagets metoder og begreber anvendes i arbejdet med konkrete problemstillinger, og det er karakteristisk for faget, at det stiller krav til samarbejdsevne, abstraktion og kreativitet.

MELLEMNIVEAU (C-niveau)

2. Undervisningsmål

Målet er, at eleverne skal
- have en datalogisk indsigt, som kombinerer praktiske færdigheder, teoretisk forståelse og generel viden
- kunne anvende et datalogisk begrebsapparat til beskrivelse og forståelse af informationsbehandling
- kunne formulere sig skriftligt og mundtligt om datalogiske emner
- kunne anvende deres kreativitet og abstraktionsevne i arbejdet med datalogiske problemstillinger
- kende til og kunne anvende datalogiske metoder i problemløsning
- kende til strukturering, formalisering og modellering og kunne anvende det på konkrete problemstillinger
- kunne implementere de udviklede modeller ved hjælp af værktøjsprogrammel og generelt programmeringssprog
- have kendskab til fundamentale algoritmer og datastrukturer
- have kendskab til eksempler på principielle begrænsninger af computerens anvendelser
- have forståelse for computerens fundamentale strukturer og processer.

3. Undervisningen

3.1 Undervisningens overordnede indhold omfatter begreberne information, struktur, proces og model. Undervisning skal dække følgende 4 områder:

Problemløsning

Undervisningen i dette område skal omfatte emnerne
- formalisering
- modelbegrebet
- algoritmer
- implementering.

Eleverne skal beskæftige sig med den arbejdsproces, der består i at strukturere og formalisere en konkret problemstilling med efterfølgende implementering i et værktøjsprogrammel eller et højere programmeringssprog.

Programmering

Undervisningen i dette område skal omfatte emnerne
- grundlæggende algoritmer
- variable og typebegrebet
- datastrukturer
- programkontrolstrukturer
- modularisering
- syntaks og semantik
- objekter, egenskaber, metoder og hændelser.

Undervisningen skal føre frem til, at eleverne kan fremstille velstrukturerede programmer i et højere programmeringssprog. Eleverne skal arbejde med en afgrænset kerne af et generelt programmeringssprog og lære syntaksen og semantikken for den pågældende kerne af programmeringssproget. Eleverne skal arbejde med datastrukturer og grundlæggende algoritmer gennem illustrative eksempler. Desuden skal eleverne stifte bekendtskab med problemstillinger, hvor en algoritmisk beskrivelse af løsningsmetoden er principiel umulig, samt eksempler, hvor problemet er praktisk uløseligt.

Maskinarkitektur

Undervisningen i dette område skal omfatte emnerne
- virtuelle maskinniveauer og systemprogrammel
- dualitet mellem program og data.

Undervisningen skal give eleverne forståelse af, hvordan en computer principielt fungerer. Der skal orienteres om de vigtigste systemprogrammer og samspillet mellem dem. Eleverne skal præsenteres for en simpel, grundlæggende model af en computer. Undervisningen skal illustrere det universelle princip bag computerens virkemåde og behandle samspillet mellem de forskellige virtuelle maskinniveauer. Eleverne skal prøve at skrive simple programmer på maskinsprogsniveau i relation til den anvendte model, og de skal opnå forståelse af dualiteten mellem program og data.

Valgfrit emne

Varighed ca. 15 timer, der skal afvikles som et særskilt forløb, der behandler et valgfrit emne. Elevernes valg af emne skal godkendes af læreren med henblik på at sikre de valgte emners faglige relevans og egnethed.

3.2 Som led i undervisningen udarbejdes 4 skriftlige rapporter. Disse skal være bredt dækkende, således at de 3 områder problemløsning, programmering og maskinarkitektur er repræsenteret. Arbejdet med hver rapport skal have et omfang svarende til 1-2 ugers undervisningstid i faget inkl. almindelig forberedelsestid. Løsning af opgaver og udarbejdelse af rapporter kan foregå i grupper på indtil 4 elever. Rapporterne rettes og kommenteres af læreren. Undervisningen omfatter i øvrigt mindre øvelsesopgaver.

3.3 Der læses 150-200 sider. De 4 rapporter indregnes med det faktiske sidetal, dog maksimalt 10 sider pr. rapport.

4. Eksamen

4.1 Der afholdes en mundtlig prøve med en forberedelsestid (inkl. instruktion og materialeudlevering) på ca. 25 min. Der eksamineres (inkl. censur) 2,5 eksaminander i timen.

4.2 Eksamensopgivelserne er på 110-130 sider inkl. mindst 3 af de 4 rapporter. De opgivne rapporter skal medregnes med det faktiske sidetal, dog maksimalt 10 sider pr. rapport. Såfremt en grupperapport indgår i eksamensopgivelserne, opgiver alle gruppedeltagerne den samlede rapport. De 3 områder problemløsning, programmering og maskinarkitektur skal være repræsenteret i

eksamensopgivelserne.

4.3 Eksamensopgivelser for selvstuderende er på ca. 200 sider. De 4 rapporter indregnes med det faktiske sidetal, dog maksimalt 10 sider pr. rapport. Som bilag til pensumindberetningen skal selvstuderende for hver rapport aflevere en beskrivelse på én side af, hvad rapporten omhandler. Pensumindberetningen for selvstuderende kan ikke godkendes, hvis disse 4 rapportbeskrivelser ikke foreligger.

4.4 Ved prøven stilles ét spørgsmål, der refererer til et stofområde/emne fra opgivelserne. Hvis eksaminanden har lavet en rapport, hvis emne ligger inden for eksamensspørgsmålet, inddrages rapporten normalt i eksaminationen. Det enkelte eksamensspørgsmål udformes med en emneoverskrift, men uddybes i øvrigt med underspørgsmål eller stikord, der kan vejlede eksaminanden. Der kan vedlægges bilag til eksamensspørgsmålet, men det skal klart fremgå, om materialet vil blive inddraget ved eksaminationen, eller om det står eksaminanden frit at benytte det. Bilagene må ikke være udformet på en sådan måde, at det er nødvendigt for eksaminanden at benytte forberedelsestiden til egentlig opgaveløsning.

4.5 Ved planlægningen af afviklingen af den mundtlige prøve fastlægger læreren efter drøftelse med holdet, om der skal benyttes computer ved prøven. Såfremt der benyttes computer, skal der være en computer til rådighed både i forberedelses- og eksaminationslokalet. Brug af computer må kun optage en mindre del af eksaminationstiden. Såvel eksaminand som eksaminator kan inddrage computeren i eksaminationen.

4.6 Der udformes normalt så mange eksamensspørgsmål til et hold, at det ikke er nødvendigt at lade enslydende spørgsmål gå igen inden for holdet, men for store hold kan det undertiden være nødvendigt at lade enkelte af spørgsmålene optræde 2 gange.

4.7 *Bedømmelseskriterier*

4.7.1 Som overordnede bedømmelseskriterier indgår, i hvilken grad eksaminanden
- kan kombinere praktiske færdigheder og teoretisk forståelse
- kan benytte et datalogisk begrebsapparat til at beskrive og forstå informationsbehandling
- har kendskab til og kan anvende datalogiske metoder
- er i stand til at formidle sin viden og tankegang om datalogiske emner
- har forståelse for computerens fundamentale strukturer og processer
- har viden om strukturering, formalisering og modellering og kan anvende begreberne
- har viden om implementering af modeller i værktøjsprogrammel og generelt programmeringssprog og kan anvende denne viden.

4.7.2 Der gives én karakter ud fra en helhedsvurdering.

Sidst redigeret den 23. juni 1999 af Område for gymnasiale uddannelser

TILBAGE TIL UNDERVISNINGSMINISTERIETS FORSIDE

B Kode

Resten af bilagene indeholder koden til LINDA programmet.

B.1 dk.dyregod.linda.Linda

```
/*
* ------------------------------------------------------------------------------

* "THE BEER-WARE LICENSE" (Revision 42):
* <doktor@dyregod.dk> wrote this file. As long as you retain this notice you
* can do whatever you want with this stuff. If we meet some day, and you
think
* this stuff is worth it, you can buy me a beer in return. Ulf Holm Nielsen
* ------------------------------------------------------------------------------

*
*/

package dk.dyregod.linda;

import org.eclipse.swt.*;
import org.eclipse.swt.custom.*;
import org.eclipse.swt.dnd.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.program.*;
import org.eclipse.swt.widgets.*;

import org.eclipse.swt.widgets.List;
import java.util.*;
import java.io.*;
import java.text.*;

import dk.dyregod.linda.util.*;

public class Linda
{
    private static final int CPU_WIDTH = 200;
    private static final int CPU_HEIGHT = 200;
    private static final int BUS_WIDTH = 300;
    private static final int BUS_HEIGHT = 250;
    private static final int INPUT_WIDTH = 75;
    private static final int INPUT_HEIGHT = 75;
    private static final int OUTPUT_WIDTH = 75;
    private static final int OUTPUT_HEIGHT = 75;

    private static final int STORAGE_WIDTH = 180;

    private static Display display;
    private static Shell shell;

    private Font font;

    private StyledText editor;
    private Table table;
    private TableEditor tableEditor;
    private Group mem;
    private Group egroup;

    private static ResourceBundle resourceBundle;

    private Maskine maskine = new Maskine(this);
    DecimalFormat form = new DecimalFormat("####");
    DecimalFormat form2 = new DecimalFormat("##");

    Text xText;
    Text yText;
    Text aText;
```

```
    Text pText;
    Text iText;
    Text adrText;
    Text ldrText;
    Text larText;
    Text ioText;

    Label status;
    String statusStr = "Klar";

    private static String splashCmd = null;

    int type = 0;
    boolean go = true;

    Thread thread;

    static String getResourceString(String key)
    {
        try
        {
            return resourceBundle.getString(key);
        }
        catch (MissingResourceException e)
        {
            return key;
        }
        catch (NullPointerException e)
        {
            return "!" + key + "!";
        }
    }

    void fill()
    {
        shell.setText(getResourceString("title"));
        shell.setImage(Images.getImage("general.icons.main"));

        shell.setMenuBar(createMenu());

        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 3;
        gridLayout.marginWidth = 0;
        gridLayout.marginHeight = 0;

        shell.setLayout(gridLayout);

        doCPU();
        /*
        Composite comp2 = new Composite(shell, SWT.BORDER);
        comp2.setLayout(new FillLayout());
        comp2.setLayoutData(new GridData(GridData.FILL_VERTICAL));
        */
        doIO();
        doMemory();



        Composite comp2 = new Composite(shell, SWT.NONE);
        comp2.setLayout(new FillLayout());
        GridData dd = new GridData(GridData.HORIZONTAL_ALIGN_CENTER |
GridData.GRAB_HORIZONTAL);
        dd.horizontalSpan = 2;
```

```
        comp2.setLayoutData(dd);
        Button input = new Button(comp2, SWT.BORDER);
        //input.setText(getResourceString("button.run.text"));
        input.setImage(Images.getImage("button.icon.run"));
        input.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                run();
            }
        });
        input = new Button(comp2, SWT.BORDER);
        //input.setText(getResourceString("button.singlestep.text"));
        input.setImage(Images.getImage("button.icon.singlestep"));
        input.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                singlestep();
            }
        });
        input = new Button(comp2, SWT.BORDER);
        //input.setText(getResourceString("button.mikrostep.text"));
        input.setImage(Images.getImage("button.icon.mikrostep"));
        input.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                mikrostep();
            }
        });
        input = new Button(comp2, SWT.BORDER);
        //input.setText(getResourceString("button.stop.text"));
        input.setImage(Images.getImage("button.icon.stop"));
        input.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                stop();
            }
        });
        input = new Button(comp2, SWT.BORDER);
        //input.setText(getResourceString("button.reset.text"));
        input.setImage(Images.getImage("button.icon.reset"));
        input.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                reset();
            }
        });
        status = new Label(shell, SWT.BORDER);
        GridData dat =
            new GridData(GridData.FILL_HORIZONTAL | GridData.HORIZONTAL_ALIGN_
FILL);
        dat.horizontalSpan = 3;
        status.setLayoutData(dat);

    }

    void open()
    {
        shell.setSize(640, 480);
        shell.setLocation(
```

```
            (display.getClientArea().width - shell.getSize().x) / 2,
            (display.getClientArea().height - shell.getSize().y) / 2);
        shell.open();

        update();
        drive();

        // Cleanup
        //workerStop();
        Images.free();
        display.dispose();
    }
    void init()
    {
        // Create the wmain dtb windows
        form.setMinimumIntegerDigits(4);
        form2.setMinimumIntegerDigits(2);

        display = new Display();
        Images.init(display);
        shell = new Shell();

        fill();
    }
    void doIO()
    {
        Group comp3 = new Group(shell, SWT.NONE);
        comp3.setText(getResourceString("io.title.text"));
        GridLayout gLayout = new GridLayout();
        gLayout.numColumns = 2;
        gLayout.marginWidth = 3;
        gLayout.marginHeight = 3;
        comp3.setLayout(gLayout);

        GridData dd1 = new GridData(GridData.HORIZONTAL_ALIGN_BEGINNING |
GridData.VERTICAL_ALIGN_BEGINNING);
        dd1.horizontalSpan = 1;
        comp3.setLayoutData(dd1);


        Label lbl = new Label(comp3, SWT.NONE);
        lbl.setText(getResourceString("io.title.text"));
        ioText = new Text(comp3, SWT.BORDER | SWT.CENTER);
        ioText.setEditable(false);
        Button input = new Button(comp3, SWT.BORDER | SWT.CENTER);
        GridData dub = new GridData(GridData.FILL_HORIZONTAL);
        dub.horizontalSpan =2;
        input.setLayoutData(dub);
        input.setImage(Images.getImage("button.icon.input"));

        input.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                ioText.setEditable(false);
                maskine.registers[maskine.IO] = Integer.valueOf(ioText.
getText()).intValue();
                if(thread==null)
                    return;
                if (thread.isAlive())
                {

                    synchronized (this)
                    {
```

```
                    maskine.wait = false;
                    notify();
                }
            }
        }
    });
}
void doCPU()
{
    Group cpu = new Group(shell, SWT.NULL);
    GridData data = new GridData(GridData.VERTICAL_ALIGN_BEGINNING);
    data.horizontalSpan = 1;
    cpu.setLayoutData(data);
    cpu.setText(getResourceString("cpu.title.text"));

    GridLayout gLayout = new GridLayout();
    gLayout.numColumns = 2;
    gLayout.marginWidth = 3;
    gLayout.marginHeight = 3;
    cpu.setLayout(gLayout);

    Label xLbl = new Label(cpu, SWT.NONE);
    xText = new Text(cpu, SWT.BORDER | SWT.CENTER);
    xText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    xLbl.setText(getResourceString("cpu.x.text"));

    Label yLbl = new Label(cpu, SWT.NONE);
    yText = new Text(cpu, SWT.BORDER | SWT.CENTER);
    yText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    yLbl.setText(getResourceString("cpu.y.text"));

    Label aLbl = new Label(cpu, SWT.NONE);
    aText = new Text(cpu, SWT.BORDER | SWT.CENTER);
    aText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    aLbl.setText(getResourceString("cpu.a.text"));

    Label pLbl = new Label(cpu, SWT.NONE);
    pText = new Text(cpu, SWT.BORDER | SWT.CENTER);
    pText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    pLbl.setText(getResourceString("cpu.p.text"));

    Label iLbl = new Label(cpu, SWT.NONE);
    iText = new Text(cpu, SWT.BORDER | SWT.CENTER);
    iText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    iLbl.setText(getResourceString("cpu.i.text"));

    Label adrLbl = new Label(cpu, SWT.NONE);
    adrText = new Text(cpu, SWT.BORDER | SWT.CENTER);
    adrText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    adrLbl.setText(getResourceString("cpu.adr.text"));

}
void doMemory()
{
    // Setup the memory group

    mem = new Group(shell, SWT.NULL);
    GridData grid =
        new GridData(
            GridData.FILL_VERTICAL
                | GridData.HORIZONTAL_ALIGN_END
                | GridData.VERTICAL_ALIGN_FILL);
    grid.verticalSpan = 2;
    mem.setLayoutData(grid);
```

```
        mem.setText(getResourceString("mem.title.text"));

        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 2;
        gridLayout.marginWidth = 3;
        gridLayout.marginHeight = 3;
        mem.setLayout(gridLayout);

        Label ldrLbl = new Label(mem, SWT.NONE);
        ldrText = new Text(mem, SWT.BORDER | SWT.CENTER);
        ldrText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        ldrLbl.setText(getResourceString("mem.ldr.text"));

        Label larLbl = new Label(mem, SWT.NONE);
        larText = new Text(mem, SWT.BORDER | SWT.CENTER);
        larText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        larLbl.setText(getResourceString("mem.lar.text"));

        table = new Table(mem, SWT.BORDER | SWT.FULL_SELECTION | SWT.SINGLE);
        GridData gridData = new GridData();
        gridData.verticalAlignment = GridData.FILL;
        gridData.horizontalSpan = 2;
        gridData.grabExcessVerticalSpace = true;
        gridData.horizontalAlignment = GridData.FILL;
        gridData.grabExcessHorizontalSpace = true;
        //gridData.verticalAlignment = GridData.GRAB_VERTICAL;
        table.setLayoutData(gridData);
        table.setHeaderVisible(true);
        table.setLinesVisible(true);

        table.setMenu(createPopUpMenu());
        tableEditor = new TableEditor(table);

        table.addSelectionListener(new SelectionAdapter()
        {
            public void widgetDefaultSelected(SelectionEvent e)
            {
                editTableCell();
            }
        });

        TableColumn column = new TableColumn(table, SWT.CENTER);
        column.setText(getResourceString("mem.add.text"));
        column.setWidth(50);
        column = new TableColumn(table, SWT.NONE);
        column.setText(getResourceString("mem.value.text"));
        column.setWidth(80);

        for (int i = 0; i < 100; i++)
        {
            TableItem item = new TableItem(table, SWT.CENTER);
            item.setText(new String[] { form2.format(i), form.format(maskine.
lager[i])});
        }
    }
    void editTableCell()
    {
        // Clean up any previous editor control
        Control oldEditor = tableEditor.getEditor();
        if (oldEditor != null)
            oldEditor.dispose();

        // Identify the selected row
```

```
    int index = table.getSelectionIndex();
    if (index == -1)
        return;
    final TableItem item = table.getItem(index);

    // The control that will be the editor must be a child of the Table
    final Text text = new Text(table, SWT.NONE);
    text.addFocusListener(new FocusAdapter()
    {
        public void focusLost(FocusEvent e)
        {
            try
            {

                int i = Integer.parseInt(text.getText());
                if (i < 10000)
                    maskine.lager[table.getSelectionIndex()] = i;
            }
            catch (NumberFormatException nfe)
            {
            }
            text.dispose();
            update();
        }
    });

    //The text editor must have the same size as the cell and must
    //not be any smaller than 50 pixels.
    tableEditor.horizontalAlignment = SWT.LEFT;
    tableEditor.grabHorizontal = true;
    tableEditor.minimumWidth = 50;

    // Open the text editor in the second column of the selected row.
    tableEditor.setEditor(text, item, 1);

    // Assign focus to the text control
    text.setFocus();

}

void close()
{
    Images.free();
    display.dispose();
}

private Menu createMenu()
{
    Menu menuBar = new Menu(shell, SWT.BAR);
    shell.setMenuBar(menuBar);

    //create each header and subMenu for the menuBar
    createFileMenu(menuBar);
    createRunMenu(menuBar);
    //createEditMenu(menuBar);
    //createSearchMenu(menuBar);
    createHelpMenu(menuBar);

    return menuBar;
}
private void createRunMenu(Menu menuBar)
{
    MenuItem item = new MenuItem(menuBar, SWT.CASCADE);
    item.setText(getResourceString("menu.run.title"));
```

```
        Menu menu = new Menu(shell, SWT.DROP_DOWN);
        item.setMenu(menu);

        MenuItem subItem = new MenuItem(menu, SWT.NULL);
        subItem.setText(getResourceString("menu.run.run.title"));
        subItem.setAccelerator(SWT.CTRL + 'G');
        subItem.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                run();
            }
        });

        subItem = new MenuItem(menu, SWT.NULL);
        subItem.setText(getResourceString("menu.run.singlestep.title"));
        subItem.setAccelerator(SWT.CTRL + 'T');
        subItem.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                singlestep();
            }
        });
        subItem = new MenuItem(menu, SWT.NULL);
        subItem.setText(getResourceString("menu.run.mikrostep.title"));
        subItem.setAccelerator(SWT.CTRL + 'M');
        subItem.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                mikrostep();
            }
        });

        subItem = new MenuItem(menu, SWT.NULL);
        subItem.setText(getResourceString("menu.run.stop.title"));
        subItem.setAccelerator(SWT.CTRL + 'H');
        subItem.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                stop();
            }
        });
        item = new MenuItem(menu, SWT.SEPARATOR);
        subItem = new MenuItem(menu, SWT.NULL);
        subItem.setText(getResourceString("menu.run.reset.title"));
        subItem.setAccelerator(SWT.CTRL + 'R');
        subItem.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {

                reset();
            }
        });
    }
    private void createHelpMenu(Menu menuBar)
    {
        MenuItem item = new MenuItem(menuBar, SWT.CASCADE);
        item.setText(getResourceString("menu.help.title"));
        Menu menu = new Menu(shell, SWT.DROP_DOWN);
        item.setMenu(menu);
```

```
        //Help -> About Text Editor
        MenuItem subItem = new MenuItem(menu, SWT.NULL);
        subItem.setText(getResourceString("menu.help.about.title"));
        subItem.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                MessageBox box = new MessageBox(shell, SWT.ICON_INFORMATION |
SWT.OK);
                box.setText(getResourceString("menu.help.about.box.title"));
                box.setMessage(getResourceString("menu.help.about.box.text"));
                box.open();
            }
        });
    }
    private Menu createPopUpMenu()
    {
        Menu popUpMenu = new Menu(shell, SWT.POP_UP);

        /**
         * Adds a listener to handle enabling and disabling
         * some items in the Edit submenu.
         */
        popUpMenu.addMenuListener(new MenuAdapter()
        {
            public void menuShown(MenuEvent e)
            {
                Menu menu = (Menu) e.widget;
            }
        });

        //Edit
        MenuItem item = new MenuItem(popUpMenu, SWT.CASCADE);
        item.setText(getResourceString("popup.edit"));
        item.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                editTableCell();
            }
        });

        new MenuItem(popUpMenu, SWT.SEPARATOR);

        //Clear this
        item = new MenuItem(popUpMenu, SWT.CASCADE);
        item.setText(getResourceString("popup.clear"));
        item.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                maskine.lager[table.getSelectionIndex()] = 0;
                update();
            }
        });

        //Clear all
        item = new MenuItem(popUpMenu, SWT.CASCADE);
        item.setText(getResourceString("popup.clearall"));
        item.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
```

```
            clearAll();
        }
    });

    return popUpMenu;
}
void clearAll()
{
    for (int i = 0; i < maskine.lager.length; i++)
    {
        maskine.lager[i] = 0;
    }
    for (int i = 0; i < maskine.registers.length; i++)
    {
        maskine.registers[i] = 0;
    }
    update();
}
void run()
{
    maskine.type = 0;
    thread = new Thread(maskine);
    thread.setPriority(Thread.MIN_PRIORITY);
    thread.start();
}
void singlestep()
{
    if (thread == null)
    {
        thread = new Thread(maskine);
    }
    maskine.type = 1;
    if (thread.isAlive())
    {

        synchronized (this)
        {
            maskine.halt = false;
            notify();
        }
    }
    else
    {
        thread = new Thread(maskine);
        thread.setPriority(Thread.MIN_PRIORITY);
        thread.start();
    }
}
void mikrostep()
{
    if (thread == null)
    {
        thread = new Thread(maskine);
    }
    maskine.type = 2;
    if (thread.isAlive())
    {

        synchronized (this)
        {
            maskine.halt = false;
            notify();
        }
    }
```

```
        else
        {
            thread = new Thread(maskine);
            thread.setPriority(Thread.MIN_PRIORITY);
            thread.start();
        }
    }
    void stop()
    {
        maskine.stop = true;
    }
    void reset()
    {
        if (thread == null)
            return;
        if (thread.isAlive())
        {

            synchronized (this)
            {
                maskine.type = 0;
                maskine.stop = true;
                maskine.halt = false;
                notify();
            }
        }
        thread = null;
        maskine.reset();
        update();
    }
    void saveFile()
    {
        final String textString;
        FileDialog fileDialog = new FileDialog(shell, SWT.SAVE);

        fileDialog.setFilterExtensions(new String[] { "*.ls", "*.*" });
        fileDialog.open();
        String name = fileDialog.getFileName();

        if ((name == null) || (name.length() == 0))
            return;

        File file = new File(fileDialog.getFilterPath(), name);
        if (file.exists())
        {
            String message =
                MessageFormat.format(
                    getResourceString("warning.file.exist"),
                    new String[] { file.getName()});
            if (!YesNoMessage.display(new Shell(), message, getResourceString(
"warning")))
                return;
        }

        try
        {
            FileOutputStream stream = new FileOutputStream(file.getPath()+".
ls");
            try
            {
                Writer out = new BufferedWriter(new OutputStreamWriter(stream)
);
                for (int i = 0; i < maskine.lager.length; i++)
                {
```

```
                    out.write(String.valueOf(maskine.lager[i]));
                    out.write("\n");
                    out.flush();
                }
                stream.close();
            }
            catch (IOException e)
            {
                String message =
                    MessageFormat.format(
                        getResourceString("error.file.ioerror"),
                        new String[] { file.getName()});
                ErrorMessage.display(new Shell(), getResourceString("error"),
message);
                return;
            }
        }
        catch (FileNotFoundException e)
        {
            String message =
                MessageFormat.format(
                    getResourceString("error.file.notfound"),
                    new String[] { file.getName()});
            ErrorMessage.display(new Shell(), getResourceString("error"),
message);
            return;
        }
    }
    void openFile()
    {
        final String textString;
        FileDialog fileDialog = new FileDialog(shell, SWT.OPEN);

        fileDialog.setFilterExtensions(new String[] { "*.ls", "*.*" });
        fileDialog.open();
        String name = fileDialog.getFileName();

        if ((name == null) || (name.length() == 0))
            return;

        File file = new File(fileDialog.getFilterPath(), name);
        if (!file.exists())
        {
            String message =
                MessageFormat.format(
                    getResourceString("error.file.noexist"),
                    new String[] { file.getName()});
            ErrorMessage.display(new Shell(), message, getResourceString("
error"));
            return;
        }

        try
        {
            FileInputStream stream = new FileInputStream(file.getPath());
            try
            {
                BufferedReader in = new BufferedReader(new InputStreamReader(
stream));
                String st;
                int i = 0;
                while (i < 100)
                {
                    st = in.readLine();
```

```
                    maskine.lager[i] = Integer.parseInt(st);
                    i++;
                }
                stream.close();
                update();
            }
            catch (IOException e)
            {
                String message =
                    MessageFormat.format(
                        getResourceString("error.file.ioerror"),
                        new String[] { file.getName()});
                ErrorMessage.display(new Shell(), getResourceString("error"),
message);
                return;
            }
        }
        catch (FileNotFoundException e)
        {
            String message =
                MessageFormat.format(
                    getResourceString("error.file.notfound"),
                    new String[] { file.getName()});
            ErrorMessage.display(new Shell(), getResourceString("error"),
message);
            return;
        }

    }
    private void createFileMenu(Menu parent)
    {
        Menu menu = new Menu(parent);
        MenuItem header = new MenuItem(parent, SWT.CASCADE);
        header.setText(getResourceString("menu.file.title"));
        header.setMenu(menu);

        MenuItem item;

        item = new MenuItem(menu, SWT.CASCADE);
        item.setText(getResourceString("menu.file.open.title"));
        item.setAccelerator(SWT.CTRL + 'O');
        item.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent event)
            {
                openFile();
            }
        });
        item = new MenuItem(menu, SWT.PUSH);
        item.setText(getResourceString("menu.file.save.title"));
        item.setAccelerator(SWT.CTRL + 'S');
        item.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
            {
                saveFile();
            }
        });
        item = new MenuItem(menu, SWT.SEPARATOR);
        item = new MenuItem(menu, SWT.PUSH);
        item.setText(getResourceString("menu.file.exit.title"));
        item.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent e)
```

```
            {
                close();
            }
        });
    }

    public void update()
    {

        shell.getDisplay().syncExec(new Runnable()
        {
            public void run()
            {
                TableItem[] items = table.getItems();
                for (int i = 0; i < items.length; i++)
                {
                    items[i].setText(
                        new String[] { form2.format(i), form.format(maskine.
lager[i])});
                }
                larText.setText(form.format(maskine.registers[maskine.LAR]));
                ldrText.setText(form.format(maskine.registers[maskine.LDR]));
                xText.setText(form.format(maskine.registers[maskine.X]));
                yText.setText(form.format(maskine.registers[maskine.Y]));
                aText.setText(form.format(maskine.registers[maskine.A]));
                pText.setText(form2.format(maskine.registers[maskine.P]));
                iText.setText(form2.format(maskine.registers[maskine.I] / 100)
);
                adrText.setText(form2.format(maskine.registers[maskine.ADR]));
                ioText.setText(form.format(maskine.registers[maskine.IO]));

                status.setText(statusStr);

                shell.redraw();

            }
        });

    }
    public Shell getShell()
    {
        return shell;
    }
    void drive()
    {
        // Event loop
        while (!shell.isDisposed())
        {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }

    public static void main(String[] args)
    {
        System.out.println("LINDA running on:");
        System.out.println(
            System.getProperty("java.vm.vendor")
                + " "
                + System.getProperty("java.version")
                + " "
                + System.getProperty("java.vm.name"));
        System.out.println(
            System.getProperty("os.name")
```

```
                + " "
                + System.getProperty("os.version")
                + " "
                + System.getProperty("os.arch"));
        System.out.println("Classpath = " + System.getProperty("java.class.
path"));
        System.out.println("Ext dirs = " + System.getProperty("java.ext.dirs")
);
        System.out.println(System.getProperty("java.class.version") + "\n");

        System.out.println(System.getProperty("user.name"));
        System.out.println(System.getProperty("user.home"));
        System.out.println(System.getProperty("user.dir"));

        System.out.println("\nstarting..");

        // Extract the command to end the splash window.
        for (int index = 0; index < args.length; index++)
        {
            System.out.println("    args[" + index + "] = '" + args[index] + "
'");
            if (args[index].equals("-endsplash") && (index + 1) < args.length)
            {
                splashCmd = args[index + 1];
            }
        }

        resourceBundle = ResourceBundle.getBundle("LindaResource");
        Linda linda = new Linda();
        linda.init();
        try
        {
            Thread.sleep(1000);
        }
        catch (Exception e)
        {
        }
        System.out.println("Initialization complete!");

        // Initialization is complete so execute the end splash command.
        if (splashCmd != null)
        {
            try
            {
                Runtime.getRuntime().exec(splashCmd);
            }
            catch (Exception e)
            {
            }
        }

        linda.open();

        try
        {
            Thread.sleep(2000);
        }
        catch (Exception e)
        {
        }
        System.out.println("Program is terminating!");

    }
}
```

B.2 dk.dyregod.linda.Maskine

```java
/*
* ------------------------------------------------------------------------

* "THE BEER-WARE LICENSE" (Revision 42):
* <doktor@dyregod.dk> wrote this file. As long as you retain this notice you
* can do whatever you want with this stuff. If we meet some day, and you
think
* this stuff is worth it, you can buy me a beer in return. Ulf Holm Nielsen
* ------------------------------------------------------------------------

*
*/

package dk.dyregod.linda;

import dk.dyregod.linda.util.*;
import java.util.BitSet;
import java.util.Hashtable;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

public class Maskine implements Runnable
{

    Linda shell;

    public final static int X = 0;
    public final static int Y = 1;
    public final static int A = 2;
    public final static int P = 3;
    public final static int I = 4;
    public final static int ADR = 5;
    public final static int LDR = 6;
    public final static int LAR = 7;
    public final static int IO = 8;

    int[] registers = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };

    String[] names = { "X", "Y", "A", "P", "I", "ADR", "LDR", "LAR", "IO" };

    int cmd;
    Hashtable instructionSet = new Hashtable();

    boolean done = false;
    boolean halt = false;
    boolean stop = false;
    boolean wait = false;

    int type = 0;

    int[] lager = new int[100];

    int bus;

    public Maskine(Linda shell)
    {
        this.shell = shell;
    }
    void setStatus(String text)
    {

        shell.statusStr = text;

    }
```

```
/*
 * update() sørger for at opdatere GUI
 *
 */

public void update()
{
    shell.update();
    if (type == 2)
    {
        halt = true;

        while (halt)
        {
            try
            {
                wait();
            }
            catch (Exception e)
            {
            }
        }

    }
    else if ((type == 1) && (done) && (!stop))
    {
        halt = true;

        while (halt)
        {
            try
            {
                wait();
            }
            catch (Exception e)
            {
            }
        }
    }
}

/*
 * run() sørger for afviklingen af program
 *
 */

public void run()
{
    stop = false;
    int i = 0;
    while (((i = executeNext()) == 0) && (!stop))
    {
        update();
    };
    if ((i == 1) && (stop))
    {

        shell.getShell().getDisplay().syncExec(new Runnable()
        {
            public void run()
            {
                MessageBox box =
                    new MessageBox(shell.getShell(), SWT.ICON_INFORMATION
```

```
| SWT.OK);
                    box.setText(shell.getResourceString("Programmet er slut"))
;
                    box.setMessage(
                        shell.getResourceString(
                            "En stop kommando er nået på adresse " + shell.
form2.format(registers[P] - 1)));
                    box.open();
                }
            });
        }

    }

    /*
     * Mikroinstruktioner
     *
     * modtagFraBus(int register)
     * ventForInput()
     * sendTilBus(int register)
     * taelOp()
     * laesLager()
     * skrivLager()
     * ALUAdd()
     * ALUSub()
     * ALUDiv()
     * ALUMult()
     *
     */

    public void modtagFraBus(int register)
    {
        registers[register] = bus;
        bus = 0;

        if (register == I)
        {
            registers[ADR] = registers[I] % 100;
        }
        setStatus("Modtag fra bus til register " + names[register]);
        update();
    }
    public void ventForInput()
    {
        setStatus("Venter på input ");
        update();
        shell.getShell().getDisplay().syncExec(new Runnable()
        {
            public void run()
            {
                shell.ioText.setEditable(true);
            }
        });
        wait = true;
        while (wait)
        {
            try
            {
                wait();
            }
            catch (Exception e)
            {
            }
        }
```

```
    setStatus("Modtag input i I/O register " + registers[IO]);
    update();
}
public void sendTilBus(int register)
{
    bus = registers[register];
    setStatus(
        "Send indholdet af "
            + names[register]
            + " ("
            + registers[register]
            + ") til bussen");
    update();
}
public void taelOp()
{
    registers[P]++;
    setStatus("Tæl P en op");
    update();
}
public void laesLager()
{
    registers[LDR] = lager[registers[LAR]];
    setStatus(
        "Læs indeholdet af adressen i LAR ("
            + registers[LAR]
            + ") ind i LDR ("
            + registers[LDR]
            + ")");
    update();
}
public void skrivLager()
{
    lager[registers[LAR]] = registers[LDR];
    setStatus(
        "Skriv indeholdet af LDR ("
            + registers[LDR]
            + ") til adressen LAR ("
            + registers[LAR]
            + ") peger på");
    update();
}
public void ALUAdd()
{
    registers[A] = registers[X] + registers[Y];
    setStatus("ALU Add (X + Y = A)");
    update();
}
public void ALUSub()
{
    registers[A] = registers[X] - registers[Y];
    setStatus("ALU Sub (X - Y = A)");
    update();
}
public void ALUDiv()
{
    registers[A] = registers[X] / registers[Y];
    setStatus("ALU Div (X / Y = A)");
    update();
}
public void ALUMult()
{
    registers[A] = registers[X] * registers[Y];
    setStatus("ALU Mult (X * Y = A)");
```

```
    update();

}

/*
 *
 * executeNext() indeholder logik til at håndtere
 * fortolkning af instruktionssættet samt mikrokode
 * til at hente næste instruktion.
 *
 */

public int executeNext()
{
    done = false;

    sendTilBus(P);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(I);
    taelOp();

    cmd = registers[I] / 100;
    switch (cmd)
    {
        case 1 :
            STOP();
            done = true;
            return 1;
        case 2 :
            HENT();
            break;
        case 3 :
            GEM();
            break;
        case 4 :
            ADD();
            break;
        case 5 :
            SUB();
            break;
        case 6 :
            MULT();
            break;
        case 7 :
            DIV();
            break;
        case 8 :
            IND();
            break;
        case 9 :
            UD();
            break;
        case 10 :
            HOP();
            break;
        case 11 :
            HNUL();
            break;
        case 12 :
            HNEG();
            break;
```

```
            default :
                shell.getShell().getDisplay().syncExec(new Runnable()
                {
                        public void run()
                        {
                            ErrorMessage.display(
                                shell.getShell(),
                                "Ukendt instruktion " + cmd + " i lagercelle " + (
registers[P] - 1),
                                "Fejl");
                        }
                    });
                    done = true;
                    stop = true;
                    return 2;

            }
            done = true;
            return 0;
        }

        /*
         * reset() sætter alle registre til 0 opdaterer GUI.
         *
         */

        public void reset()
        {
            for (int i = 0; i < registers.length; i++)
            {
                registers[i] = 0;
            }
            update();
        }

        /*
         * Instruktionssættet, en metode pr instruktion håndteres af executeNext()
         *
         */

        public void STOP()
        {
        }
        public void HENT()
        {
            sendTilBus(ADR);
            modtagFraBus(LAR);
            laesLager();
            sendTilBus(LDR);
            modtagFraBus(A);
        }
        public void GEM()
        {
            sendTilBus(ADR);
            modtagFraBus(LAR);
            sendTilBus(A);
            modtagFraBus(LDR);
            skrivLager();
        }
        public void ADD()
        {
            sendTilBus(A);
            modtagFraBus(X);
            sendTilBus(ADR);
```

```
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(Y);
    ALUAdd();
}
public void SUB()
{
    sendTilBus(A);
    modtagFraBus(X);
    sendTilBus(ADR);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(Y);
    ALUSub();
}
public void MULT()
{
    sendTilBus(A);
    modtagFraBus(X);
    sendTilBus(ADR);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(Y);
    ALUMult();
}
public void DIV()
{
    sendTilBus(A);
    modtagFraBus(X);
    sendTilBus(ADR);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(Y);
    ALUDiv();
}
public void IND()
{
    ventForInput();
    sendTilBus(IO);
    modtagFraBus(LDR);
    sendTilBus(ADR);
    modtagFraBus(LAR);
    skrivLager();
}
public void UD()
{
    sendTilBus(ADR);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(IO);
}
public void HOP()
{
    sendTilBus(ADR);
    modtagFraBus(P);
}
public void HNUL()
{
    if (registers[A] == 0)
```

```
            {
                sendTilBus(ADR);
                modtagFraBus(P);
            }
        }
        public void HNEG()
        {
            if (registers[A] < 0)
            {
                sendTilBus(ADR);
                modtagFraBus(P);
            }
        }

}
```

B.3 dk.dyregod.linda.Images

```
/*
* ------------------------------------------------------------------------

* "THE BEER-WARE LICENSE" (Revision 42):
* <doktor@dyregod.dk> wrote this file. As long as you retain this notice you
* can do whatever you want with this stuff. If we meet some day, and you
think
* this stuff is worth it, you can buy me a beer in return. Ulf Holm Nielsen
* ------------------------------------------------------------------------

*
*/

package dk.dyregod.linda;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;

import java.util.*;
import java.io.*;

import dk.dyregod.linda.util.*;

public class Images
{

    private static Hashtable table;
    private static ResourceBundle resourceBundle;

    static void init(Display display)
    {
        resourceBundle = ResourceBundle.getBundle("images");
        table = new Hashtable();

        String key;
        String val;
        Enumeration enum = resourceBundle.getKeys();
        while (enum.hasMoreElements())
        {
            key = (String) enum.nextElement();
            try
            {
                val = resourceBundle.getString(key);
                table.put(key, createBitmapImage(display, val));
            }
            catch (Exception ex)
            {
                ErrorMessage.display(new Shell(), "FEJL", "Røv og nøgler");
                ex.printStackTrace();
            }

        }
    }
    static Image createBitmapImage(Display display, String fileName)
        throws Exception
    {

        ImageData source = new ImageData(Images.class.getResourceAsStream(
fileName));
        ImageData mask = source.getTransparencyMask();
        return new Image(display, source, mask);
    }

    static void free()
```

```
    {
        if (table != null)
        {
            for (Enumeration it = table.elements(); it.hasMoreElements();)
            {
                Image image = (Image) it.nextElement();
                image.dispose();
            }
        }
    }
    static Image getImage(String key)
    {
        return (Image) table.get(key);
    }
    public static void main(String[] args)
    {
        init(null);
    }
}
```

B.4 dk.dyregod.linda.util.ErrorMessage

```java
/*
* ----------------------------------------------------------------------------

* "THE BEER-WARE LICENSE" (Revision 42):
* <doktor@dyregod.dk> wrote this file. As long as you retain this notice you
* can do whatever you want with this stuff. If we meet some day, and you
think
* this stuff is worth it, you can buy me a beer in return. Ulf Holm Nielsen
* ----------------------------------------------------------------------------

*
*/

package dk.dyregod.linda.util;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

public class ErrorMessage
{

    public static void display(Shell shell, String msg, String text)
    {
        MessageBox box = new MessageBox(shell, SWT.ICON_ERROR);
        box.setMessage(msg);
        box.setText(text);
        box.open();
    }
    public static void main(String[] args)
    {
    }
}
```

B.5 dk.dyregod.linda.util.YesNoMessage

```java
/*
* ------------------------------------------------------------------------

* "THE BEER-WARE LICENSE" (Revision 42):
* <doktor@dyregod.dk> wrote this file. As long as you retain this notice you
* can do whatever you want with this stuff. If we meet some day, and you
think
* this stuff is worth it, you can buy me a beer in return. Ulf Holm Nielsen
* ------------------------------------------------------------------------

*
*/

package dk.dyregod.linda.util;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

public class YesNoMessage
{

    public static boolean display(Shell shell, String msg, String text)
    {
        MessageBox box = new MessageBox(shell, SWT.ICON_QUESTION | SWT.YES |
SWT.NO);
        box.setMessage(msg);
        box.setText(text);
        int result = box.open();
        if (result == 64)
            return true;
        else
            return false;
    }
    public static void main(String[] args)
    {
    }
}
```

C Lærermateriale

Lærevejledning

- en introduktion til maskinarkitektur

faraz@butt.dk — **Faraz Butt**
mads@danquah.dk — **Mads Danquah**
doktor@dyregod.dk — **Ulf Holm Nielsen**

Roskilde Universitetscenter
Naturvidenskabelig Basisuddannelse Hus 14.2
3. semester

Indhold

Kapitel 1

Bekendtgørelsen og formelle krav

Undervisningsmaterialet dækker et område svarende til det af gymnasiebekendtgørelsen påkrævede for undervisning i maskinarkitektur for mellemniveau (C-niveau) datalogi. Ifølge gymnasiebekendtgørelsen inkludere det:

- virtuelle maskinniveauer og systemprogrammel
- dualitet mellem program og data.

Ud over det følger undervisningsforløbet også kravene for undervisningsmål. Netop for at holde sig til kravene for undervisningsmål, er det vigtigt at huske at eleven selv skal være i stand til at løse opgaverne på egen hånd, via. datalogiske metoder ved undervisningens afslutning.

Kapitel 2

Undervisningsforslag

Det må forventes at eleven særligt i starten af undervisningsforløbet vil befinde sig på det pre/konkret-operationelle stadie(Piaget), samt 2-3 niveau(Bloom). Hvilket betyder at de er ved at lære noget nyt, og prøver at forstå det. Derfor er det vigtigt at lade eleverne anvende deres nye viden. Timerne bør derfor bære mere præg af at være eksperimenterende, end belærende.

En måde at opnå dette er at lade eleverne sidde i grupper af 2-3 ved hver computer, og lade dem løse opgaver i fællesskab. Det er vigtigt at eleverne i de enkelte grupper ikke ligger langt fra hinanden niveau-mæssigt[1].

2.1 Gruppearbejde

Gruppearbejde kan være et særdeles effektivt middel til undervisning, dog er det til dette forløb kun nødvendigt at arbejde i små grupper, eller samarbejde med hele klassen. Det foreslås at eleverne sidder 2-3 personer pr. computer, således at de sammen kan arbejde sig frem til en løsning på problemet.

2.2 Forelæsninger

Det er vigtigt at huske at selvom forelæsninger er en meget let måde at formidle viden på , så er det stadigvæk den mest passiverende form for undervisning for eleverne. Sørg derfor for at holde elevernes opmærksomhed fanget, under hele forelæsningen. Eleverne lære ikke noget hvis de ikke finder det interessant, så derfor er det vigtigt at gøre forelæsningerne interessante. En ide kunne f.eks. være at starte forløbet med at få eleverne til at komme med deres bud på hvilke grundkomponenter en computer er

[1] Se evt ”bubblesort” for et eksempel på hvordan man kan sortere eleverne efter niveau

bygget op af. På den måde engagere man eleverne fra starten, og de har lige pludselig en interesse i at følge med i timen, frem for at falde i søvn.

2.3 Opgaver

De medfølgende opgaver er beregnet på at eleverne skal forsøge at løse dem i grupper af 2-3 personer. Nogle af opgaverne er så svære at de bør gennemgås på tavle og andre i fællesskab i klassen.

2.4 Lege

Lege, og anden fysisk aktivitet kan været et godt middel til at aktivere eleverne. Det kan ligefrem være en fordel at klargøre ting for eleverne på denne måde frem for en teoribog.

2.4.1 Bubblesort

Denne leg kan bruges til at opdele elever i små grupper.

Lad eleverne stille sig op på en række, derefter skal de afgøre om de har højere eller lavere niveau end deres sidekammarat, hvis de har lavere skal de skifte plads imod venstre, ellers imod højre. Tilsidst kan man så ”knække” rækken på midten, og lade de to rækker stille sig ved siden af hinanden. Man har nu et antal par, hvor der er en klar niveauforskel, dog uden at den bedste kommer sammen med den dårligste.

2.4.2 En elev-datamat

Eleverne skal i denne leg agere komponenter i en computer. På forhånd laves der en række små kort med beskrivelse af hvordan komponentet fungere, samt en række simple programmer. Eleverne skal nu trække et beskrivelses-kort hver. Beskrivelser kunne f.eks. være:

- Bus: Du kan modtage instruktioner eller data, og videregive den til en anden komponent. Du kan ikke ændre på noget.
- Alu: Du kan udføre aritmetriske operationer (addition, subtraktion, multiplikation og division). Du kan udføre disse operationer på indholdet af register x og y, og lægge resultatet i a. Du kan kun tilgå disse registre via bussen.
- Lager: Du kan modtage eller sende indholdet af specifikke celler. Du må ikke selv bestemme hvilke celler der tilgås, men må kun tilgå de celler du får besked på fra bussen.

Gruppen får nu udleveret et simpelt program, og får til opgave at afvikle det korrekt. Hvert medlem må kun udføre sin egen opgave, dog kan det være en ide at lade hele gruppen fungere som styreenhed.

Legens formål er at give eleverne en ide om hvordan komponenterne i en computer arbejder sammen, og kan forhåbentligt være med at give eleven en bedre forståelse.

Kapitel 3

Opsætning af ”LINDA”

3.1 Maskinkrav

LINDA er testet på Windows og Linux (testet på Intel ia32, men Intel ia64 og PPC platforme burde også være ok). Maskinkravene er ens for begge operativsystemer.

- Mininum hvad der svarer til en 166 MHz Pentium
- Ca. 26 Mb diskplads til LINDA inkl kildekode + 50 Mb til Java Development Kit
- Anbefalet minimum 64 Mb RAM, men 32 Mb kan gøre det.

3.2 Installation

LINDA kræver ingen installation. Alt hvad der er nødvendigt for at køre programmet ligger på cd’en og programmet kan også køres direkte fra cd’en. Dog er det nødvendigt at kopiere hele linda kataloget over på en eller anden form for medie med skriveadgang hvis man ønsker at rette i koden selv. Desuden er det nødvendigt at have et Java Development Kit installeret hvis man ønsker at kompilere kildekoden. På LINDA cd’en ligger Java Development Kit 1.3.1 til Windows fra Sun.

Hvis hovedformålet med at bruge LINDA er at eleverne skal kunne se hvordan det er implementeret anbefaler vi at hele projektet bliver importeret til et IDE eller at eleverne har adgang til en god editor. Der er på LINDA cd’en lagt to IDE’er; Eclipse fra IBM og Netbeans fra SUN. Begge IDE’er er open-source og derfor gratis, og kan frit benyttes.

For at kunne arbejde med kildekoden eller afvikle programmet uden brug

af exe filen, skal man være opmærksom på at programmet er afhængig af følgende filer: swt.jar skal være i classpath både ved kørsel og kompilering. Swt-win32-2019.dll skal placeres i Java Runtime Enviroments /bin katalog.

3.3 Problemer?

Hvis LINDA ikke starter kan man prøve at starte programmet med ”linda.exe -debug”. Programmet skulle da gerne skrive nogle informationer på skærmen.

Hvis ikke det er til nogen hjælp kan man sende en email til Ulf Holm Nielsen på doktor@dyregod.dk.

Kapitel 4

Opgaveløsninger

Øvelse 2.1

Det revidere program bør se således ud:

Adresse	Indhold
00:	0205
01:	0406
02:	0407
03:	0308
04:	0100
05:	0023
06:	0016
07:	0065
07:	0000

Øvelse 2.2

Programmet indlæser tre tal fra brugeren og tager gennemsnittet.

Øvelse 2.3

Opgaven er klart sværere end de andre opgaver i hæftet. Men ikke umulig. En løsning kan se således ud:

Programmet indlæser et andet program via IND instruktionen og gemmer det i lageret. Det indlæste program afvikles når brugeren indtaster 9999. Brugeren skal selv sørge for at placere en stop kommando!

Programmet kan ikke indlæse programmer der har brug for data fra lageret fra første instruktion. Det indlæste program bør gøre brug af IND instruktionen til at få data programmet skal afvikles med.

F.eks. vil følgende program ikke egne sig:

```
HENT   04
ADD 05
GEM 06
```

Det vil derimod følgende:

```
IND 05
IND 06
HENT   05
ADD 06
GEM 07
```

Programmet til indlæsning kan ses på næste side.

Man angiver i celle 99 hvor det indlæste program skal indlæses til (start adresse).Celle 98 er en tom GEM instruktion som vi skal bruge senere. Celle 97 bruges til at opbevare de indlæste data. Celle 96 indeholder brugerens stop kode: her 9999. Celle 95 bruges som peger, dvs den tælles op efterhånden som fylder lageret op ved indlæsning. Celle 94 indholder tallet 1 da vi skal tælle 95 op en ad gangen.

Adresse	Indhold	Kommentar
00:	HENT 19	Henter indholdet af adresse 19 som er instruktionen der sørger for at hoppe til det indlæste program
01:	ADD 99	Lægger adresse hvorpå brugerprogrammet til. Derved kan vi hoppe til den rigtige adresse.
02:	GEM 19	Gemmer den nye instruktion
03:	HENT 99	Henter start adressen for brugerens program
04:	GEM 95	Gemmer i 95 så vi kan tælle op
05:	HENT 95	Henter 95 igen da det er her vi hopper tilbage til.
06:	ADD 98	Lægger 98 til (gem instruktionen) således at vi kan gemme det indlæste på den adresse 95 peger på
07:	GEM 17	Gemmer den nye gem instruktion
08:	HENT 95	Henter adressen i 95
09:	ADD 94	Lægger 1 til
10:	GEM 95	Gemmer i 95
11:	IND 97	Indlæser linie af brugers program
12:	HENT 97	Henter 97
13:	SUB 96	Trækker 9999 fra
14:	HNUL 19	Hvis resultatet er 0 betyder det at brugeren indtastede 9999 og han ønsker at afslutte og vi hopper til 19
15:	HENT 97	Ellers henter vi 97
16:	ADD 99	Lægger 99 dvs start offset i lageret til således at brugerens adressering passer
17:	GEM 00	Data gemmes til adressen specificeret tidligere.
18:	HOP 05	Løkken gentages
19:	HOP 00	Hopper til brugerens program
20:	TOM 00	
—		Fortsætter med tomme celler indtil..
94:	TOM 01	
95:	TOM 00	
96:	9999	
97:	TOM 00	
98:	GEM 00	Gem ”skabelon”
99:	TOM 21	Offset værdi

Øvelse 3.1

Se Maskine.java filen i source/dk/dyregod/linda kataloget.

Øvelse 3.2

```
public void DIV()
{
    sendTilBus(A);
    modtagFraBus(X);
    sendTilBus(ADR);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(Y);
    ALUDiv();
}
```

Øvelse 3.3

Der er mange mulige måde at lave en sådan metode på. En kunne se ud som herunder. Men eleverne er ganske givet opfindsomme nok til at deres løsning ikke ser sådan ud.

```
public void KVA()
{
    sendTilBus(ADR);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(A);
    sendTilBus(A);
    modtagFraBus(X);
    sendTilBus(A);
    modtagFraBus(Y);
    ALUMult();
}
```

Og ændringen til executeNext():

```
case 13 :
    KVA();
    break;
```

D Elevmateriale

Computerarkitektur

- en introduktion til computerarkitektur med LINDA

faraz@butt.dk — **Faraz Butt**
mads@danquah.dk — **Mads Danquah**
doktor@dyregod.dk — **Ulf Holm Nielsen**

Roskilde Universitetscenter
Naturvidenskabelig Basisuddannelse Hus 14.2
3. semester

Forord

Dette hæfte er ment som en introduktion i maskinarkitektur til datalogiundervisningern i gymnasiet. Hæftet kan bruges selvstændigt eller i sammenhæng med det medfølgende program LINDA. Arbejdet med det tilhørende undervisningsforløb opfylder bekendtgørelsens krav om maskinarkitektur. Materialet forudsætter et begrænset kendskab til Java.

Materialet er lavet i forbindelse med et 3. semester projekt på Naturvidenskabelig Basisuddannelse på RUC.

I forbindelse med udarbejdelsen af dette materiale har Thorkild Skjeldborg været en stor hjælp.

Hvis du har kommentarer til materialet så send en mail til Ulf Holm Nielsen på doktor@dyregod.dk eller til Mads Danquah på mads@danquah.dk.

København, januar 2001

Faraz Butt
Mads Danquah
Ulf Holm Nielsen

Indhold

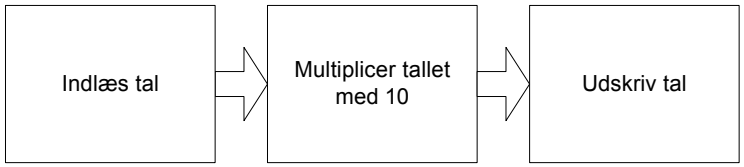
Kapitel 1

En simpel computer og hvordan den virker

Vi vil i det følgende kapitel gennemgå en simpel computermodel, der kun består af de mest basale komponenter. Desuden vil vi gennemgå, hvordan de enkelte komponenter virker.

1.1 Introduktion

Moderne computere, og for den sags skyld også umoderne, fungerer ved, at nogle elektroniske kredsløb afvikler en serie instruktioner i en bestemt rækkefølge. Ideen om en maskine, der afvikler en serie instruktioner, stammer tilbage fra 1834, hvor Charles Babbage lavede en såkaldt analytisk maskine. I 1834 var der ikke noget, der hed elektroniske kredsløb, og maskinen var derfor rent mekanisk.



Figur 1.1: En serie af instruktioner

Det var stort set umuligt at lave mekanisk, men i 1940'erne genoptog man ideen. Da havde man udviklet elektroniske kredsløb. En af de første egentlige computere, blev lavet af von Neumann i USA. Arkitekturen bag maskinen er den der stadig bruges, og den er siden blevet kaldt for von Neumann

arkitektur[1].

1.2 En simpel model

Et computerprogram består altså af en række instruktioner, der udføres i en bestemt rækkefølge. Disse instruktioner arbejder med nogle data i form af tal. For at opbygge en computer må vi have et komponent der kan forstå og udføre instruktioner. Vi skal også bruge en del, der kan indeholde instruktioner og data. Denne del skal blot indeholde tal, da både instruktioner og data er tal.

En basal **von Neumann** maskine består af følgende:

- Et lager til opbevarering af instruktioner og data.
- En kontrolenhed til at hente instruktioner og data fra lageret.
- En regneenhed til at udføre alle regneoperationer.
- Indlæsning- og udskrivningsdel, der kan skrive på skærmen eller registrere et tastetryk.

Kontrolenheden og regneenheden er næsten altid bygget sammen og kaldes en processor eller en CPU (**C**entral **P**rocessing **U**nit). Det er processoren, der afvikler instruktioner.

Vi mangler en del, der kan sørge for, at få data og instruktioner frem og tilbage mellem processoren, lageret og indlæsnings/udskrivnings-delen. Denne del kalder man en bus.

1.3 Komponenterne

1.3.1 Lager

Lageret er, hvor maskinen opbevarer data. Man kan forestille sig lageret som en masse små kasser, der hver kan indeholde et tal. Man kalder disse ”kasser” for celler. Vores maskine består af 100 celler, nummereret fra 0 til 99. En celles nummer kaldes også for dens adresse. Cellerne kan indeholde heltal mellem -9999 og 9999. Rigtige maskiner indeholder flere millioner celler, som kan indeholde tal helt op til 4 mia.

[1] Andre arbejdede med samme ide, men der var ikke mange der kendte til deres arbejde på den tid

1.3.2 Bus

En bus sørger for, at alle komponenterne i maskinen kan snakke sammen. Al data, der skal fra et komponent til et andet, skal over bussen.

1.3.3 Processor

Processoren er ”hjernen” i computeren, idet det er processoren der styrer afviklingen af programmer. Processoren indeholder to dele: en styreenhed og en regneenhed.

Regneenheden

Regneenheden, kaldes også en ALU (**A**rithmetic and **L**ogic **U**nit), udfører alle regneoperationer og gemmer resultatet i register A (A for akkumulator). Et register er det samme som en celle, men kaldes altså et register, når det ikke drejer sig om lageret. Et register kan da også ligesom hver af lagercellerne indeholde heltal fra -9999 til 9999. ALU’en kan i vores tilfælde gange, dividere, addere og subtrahere heltal. Normalt vil en ALU også kunne håndtere komma tal, men det er langt mere kompliceret og vil ikke blive behandlet yderligere.

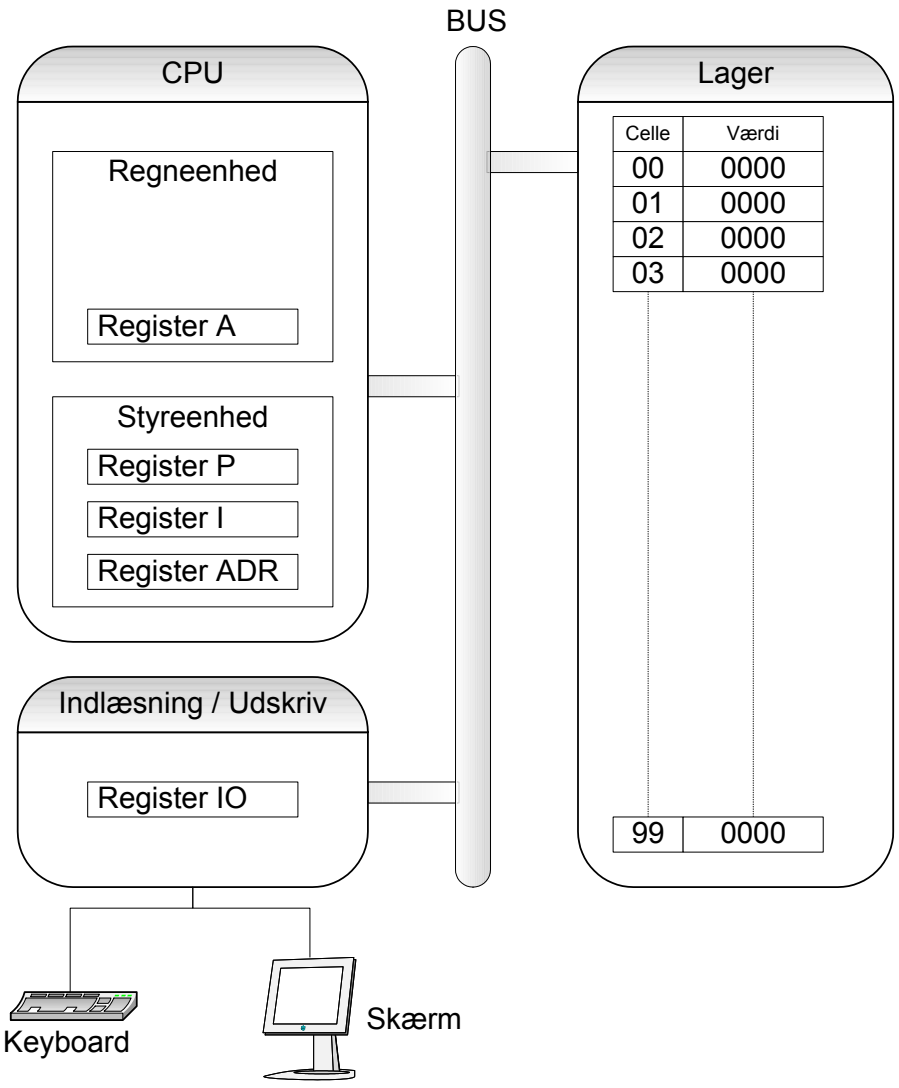
Styreenhed

Styreenheden sørger for at holde styr på, hvor i afviklingen programmet man er nået. Styreenheden har et P register (P for peger), der ”peger” på den adresse hvor den næste instruktion er placeret. Desuden har den et I register (I for instruktion), der indeholder instruktionskoden samt et ADR register (ADR for adresse), der indeholder den adresse, instruktionen skal behandle.

Indlæsning og udskrivning

Indlæsning og udskrivning er computerens eneste måde, at kommunikere til omverden på. Indlæsning til en computer, kan foregå via et tastatur, en mus, tale eller et lagermedie som f.eks. en CD. Udskrivning kan foregå til en skærm, et stykke papir eller en række lamper som på en lystavle. I vores maskine er den eneste mulighed for indlæsning og udlæsning **I/O** registret (**I**nput/**Output**). Når data flyttes til I/O registret, bliver det vist for brugeren. Hvis computeren beder om input fra brugeren, vil brugerens input det blive overført til I/O registret, hvorefter det kan viderebehandles.

Vi kan nu lave et diagram over computermodellen. Resultatet kan ses på figur 1.2 side 4.



Figur 1.2: Computermodellen

Kapitel 2

Programmering af computeren

Her vil vi gennemgå hvordan man programmerer maskinen vi lærte om i sidste kapitel.

2.1 Maskinniveauer

Man kan opdele en computer i flere niveauer; traditionelt opdeles de i følgende:

Niveau 6: Bruger programmer, som f.eks. tekstbehandling eller lign.

Niveau 5: Programmeringssprog

Niveau 4: Assemblersprog

Niveau 3: Operativsystem

Niveau 2: Maskinkode

Niveau 1: Mikroprogram

Niveau 0: Digitale kredsløb

Hvert niveau er bygget v.ha. af det foregående, således består maskinkode af mikroprogrammer, som igen bygger oven på den funktionalitet, som de digitale kredse har.

Hvis man ønsker at lave et program til vores simple computer, må man indtil videre, gøre det i maskinkode, dvs niveau 2.

2.2 Maskinkode

En computer forstår på maskinkode niveau kun meget få instruktioner. Det kan f.eks. være "`flyt data fra lagercelle 8 til register A`". Denne instruktion kalder vi for `HENT` efterfulgt af adressen, hvor der skal hentes fra. Instruktionen skal desuden ligge i lageret, hvor der jo kun er plads til 4 cifrede tal. Derfor oversætter vi instruktionen til et tal mellem 0 og 99, så den kan ligge i de to første cifre af lagercellen. De sidste to cifre bruger vi til den adresse, der skal hentes fra. En liste over alle instruktioner kan ses i bilaget.

Generelt vil alle instruktioner bortset fra `STOP`, der stopper afviklingen af programmet, skulle bruge en adresse i lageret, som de skal udføre operationen på.

Eksempel 2.1

For nemheds skyld bruger vi symbolske betegnelser for alle instruktionerne og først, når de skal indtastes i lageret, "oversætter" vi dem til tal. Et simpelt program, der adderer to tal, begge gemt i lageret på henholdsvis adresse 3 og 4, ser således ud i symbolsk maskinkode:

```
00: HENT 03
01: ADD  04
02: STOP 00
03: TOM  07
04: TOM  05
```

I lageret vil det se sådan ud:

Adresse	Indhold
00:	0203
01:	0404
02:	0100
03:	0007
04:	0005

Programmet henter indholdet af cellen med adresse 03 (her er indholdet 7) til register A. Derefter adderes indholdet af cellen med adresse 04 (her er indholdet 5) til A registret, og resultatet gemmes i A registret (som så er lig 5 + 7 = 12) og overskriver altså den gamle værdi.

Da lageret indeholder både instruktioner og data, og da computeren ikke kan skelne mellem de to, er det op til programmøren at skelne og undgå

fejltagelser. Derfor er det nødvendigt at indsætte en `STOP` kommando, når programmet ikke må køre længere.

2.2.1 Instruktioner

Da vores computer blot er en model, har den også et ret begrænset instruktionssæt, 13 instruktioner ialt. Moderne computere som f.eks. en fra Apple eller en af de utallige PC producenter har langt større instruktionssæt og mange flere registre. Alle de 13 instruktioner, som vores computer, forstår kan ses i oversigten i bilaget.

Instruktionerne i vores computer kan opdeles i 5 kategorier:

Kontrolinstruktioner

Vi har i maskinen kun en instruktion til at styre afviklingen af programmet; det er `STOP`, som vi allerede har omtalt.

Regneinstruktioner

Regneinstruktioner i vores maskine udgøres af `ADD`, `SUB`, `DIV` og `MULT`, som kan henholdsvis addere, subtrahere, dividere og multiplicere.

Flytteinstruktioner

De to flytteinstruktioner i maskinen er `HENT` og `GEM`. De henholdsvis henter og gemmer til og fra register A. Adressedelen af instruktionen angiver hvilke celle der skal læses/skrives til.

Hopinstruktioner

Det er muligt at hoppe til en ny adresse, dvs angive, at næste instruktion, der skal afvikles skal hentes fra den angivne adresse. Der findes tre forskellige hopinstruktioner; to betingede og en ubetinget. den ubetingede `HOP` sætter blot P registret til adressedelen. De to betingede `HNUL` og `HNEG` hopper kun, hvis A registret er lig 0 eller for `HNEG`, at A registret er negativt.

Indlæsnings- og udlæsningsinstruktioner

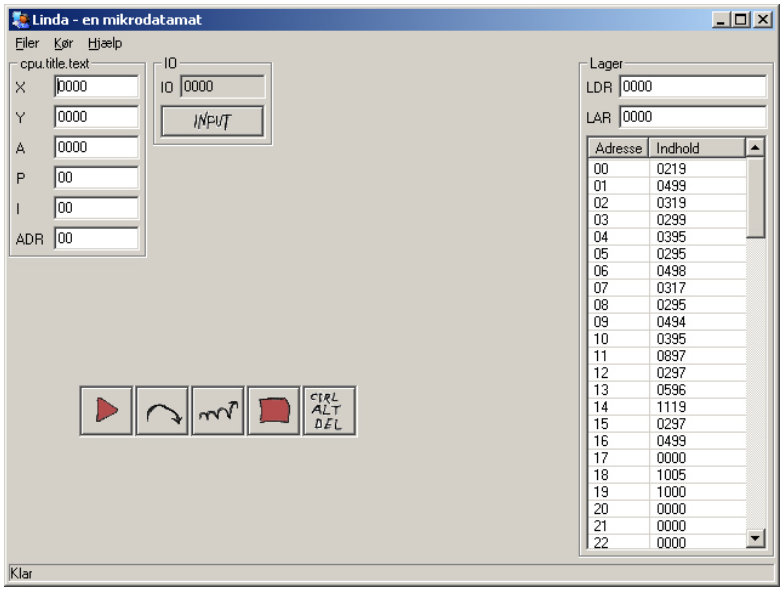
Der er en enkelt instruktion til indlæsning og en enkelt til udlæsning. `IND` venter på input fra brugeren og gemmer værdien på adressen angivet i adresse delen. `UD` skriver værdien på adressen i adressedelen til I/O registret.

2.3 Maskinkode i LINDA

Til dette hæfte hører også et program kaldet LINDA[1]. LINDA er et program til at simulere computermodellen i dette hæfte.

2.3.1 Indtastning af maskinkode i LINDA

Når programmet startes fremkommer følgende skærmbillede:



Figur 2.1: LINDA

Tabellen i højre side repræsenterer lageret. Det er muligt v.ha. dobbelt eller højre-klik at ændre indholdet af den enkelte celle. Når programmet er indtastet, er der mulighed for at afvikle programmet enten fra ende til anden eller en instruktion af gangen. Det er henholdsvis knappen længst til venstre og den næste i rækken. De er også tilgængelige fra ”Kør”-menuen. De to grupper af tekstbokse repræsenterer de registre forskellige registre i computeren.

Det er i LINDA muligt at gemme sit program, dette gøres ved at trykke gem i fil menuen. Åben henter et gemt program ind igen og overskriver hele lageret.

Tip Det er en god ide først at lave programmet på papir og vente med ind-

[1]Hvis ikke programmet fulgte med dette hæfte kan du hente det på http://www.dyregod.dk/linda

tastningen, til man ved hvor meget, det kommer til at fylde. Det er rigtig surt at have glemt i linie i starten af programmet.

2.3.2 Afvikling af programmer i LINDA

Ved afvikling af `IND` instruktionen vil det være muligt at indtaste en værdi i tekstboksen IO. Værdien indlæses først, når brugeren trykker på ”Input”-knappen.

Nulstil-knappen og -menupunktet sætter alle registre til nul, således kan maskinen afvikle programmet igen.

2.4 Øvelser

Øvelse 2.1

Indtast programmet fra eksempel 2.1, der adderer to tal, i LINDA og gå igennem programmet med enkelttrin.

Lav nu programmet så det istedet for at adderer to tal adderer tre tal. Resultatet skal gemmes på lageradresse 08.

Øvelse 2.2

Oversæt følgende program til maskinkode og indtast det i LINDA. Gå igennem programmet med enkelttrin.

Adresse	Indhold	
00:	IND	10
01:	IND	11
02:	IND	12
03:	HENT	10
04:	ADD	11
05:	ADD	12
06:	DIV	13
07:	GEM	9
08:	STOP	
09:	TOM	00
10:	TOM	00
11:	TOM	00
12:	TOM	00
13:	TOM	3

Hvad gør programmet?

Øvelse 2.3 - Svær

Lav et program der v.ha. `IND` instruktioner får brugeren til at indtaste et program. Når brugeren indtaster 9999 stopper indlæsningen og det indtastede program afvikles.

Kapitel 3

Mikrokode

Vi har i det foregående afsnit set på, hvordan man programmerer i maskinkode med LINDA. I det følgende vil vi bevæge os et niveau længere ned. Vi vil kigge på, hvordan man opbygger hver enkelt instruktion af mikrokode.

3.1 En udvidet computermodel

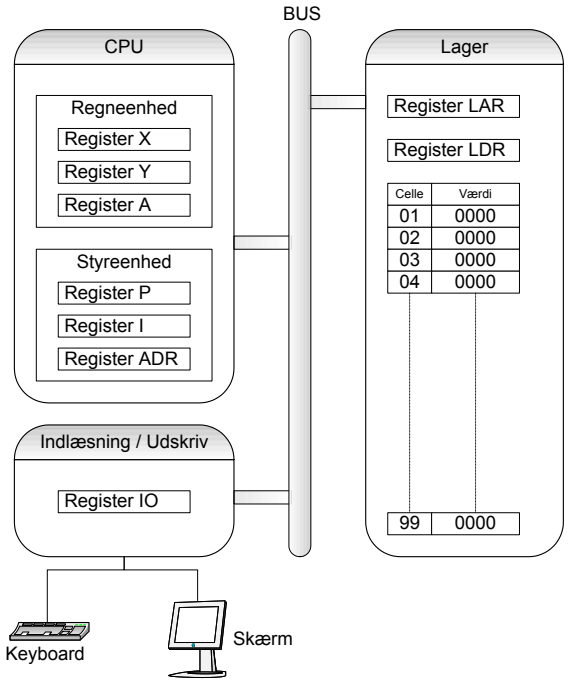
Men før vi beskæftiger os mere med mikrokode, bliver vi lige nødt til at udvide vores computermodel. Modellen, som blev introduceret i kapitel 2 skjuler nogle ting, som ikke var nødvendige at vide da vi lavede programmer på maskinniveau.

Vi vil nu se lidt nærmere på, hvad der egentlig sker, når man henter indholdet af en celle, og når man lægger to tal sammen i ALU'en.

I virkeligheden indholder lageret to registre: **LAR** og **LDR**, som står for henholdsvis **L**ager **A**dresse **R**egister og **L**ager **D**ata **R**egister. Desuden har man fra processoren ikke direkte adgang til lageret, og al kommunikation mellem processor og lager skal gå igennem bussen til de to registre LAR og LDR. Når man ønsker at hente indholdet af en celle via `HENT` instruktionen, sættes LAR til adressen på cellen. Derefter sendes besked om at man ønsker at lageret læser indholdet fra cellen, hvis adresse er angivet i LAR, ind i LDR. Derefter kan register A sættes lig register LDR. `GEM` instruktionen fungerer ligesådan. Blot bagvendt.

ALU'en indeholder ud over A registret to registre til, som kun bruges midlertidigt under regneoperationer, disse er register X og register Y. Ønsker vi f.eks. at udføre instruktionen `ADD`, starter maskinen med at overføre indholdet af register A til register X. Derefter følger den samme fremgangsmåde som ved `HENT`, blot med den forskel at indholdet af LDR lægges over i Y.

Derefter sørger elektroniske kredsløb i ALU'en for at lave de egentlige aritmetiske operationer. Og resultatet placeres i register A. Princippet er det samme for alle de aritmetiske operationer.



Figur 3.1: Den udvidede computermodel

3.2 Introduktion til mikrokode

Som nævnt i sidste kapitel så består hver instruktion af en række mikroinstruktioner. Computerens mikroinstruktioner er endnu mere begrænsede end de almindelige instruktioner, og der går flere mikroinstruktioner til at opbygge en enkelt instruktion.

Basalt set kan vores computer kun meget få ting. Den kan regne i ALU'en. Den kan flytte data mellem registre og lageret og eksterne enheder. Mikroinstruktionerne har ingen anden funktion end det komponenterne i computeren stiller til rådighed. Vi har derfor følgende instruktioner til rådighed:

Læs lager - Læser adressen i LAR og lægger indholdet i LDR

Skriv lager - Skriver indholdet af LDR på adressen i LAR

Send til bus - Sender indholdet af et register til bussen

Hent fra bus - Flytter indholdet af bussen til et register

Vent på input - Venter til IO registret er blevet sat af brugeren

Tæl op - lægger 1 til p registret

ALU Addition - adderer tallene i X og Y registrene sammen og lægger resultatet i A registret

ALU Subtraktion - subtraherer Y fra X og lægger resultatet i A

ALU Division - Dividerer X med Y og lægger resultatet i A

ALU Multiplikation - Multiplicerer X med Y og lægger resultatet i A

Det er disse mikroinstruktioner, alt i computermodellen skal opbygges af. Hvis vi bruger eksemplet fra tidligere, hvor vi ønsker at lægge to tal sammen:

HENT 03
ADD 04

Alle instruktioner har en del mikrokode til fælles. Koden til at hente en instruktion fra lageret er den samme hver gang. For at hente en instruktion fra lageret v.ha. mikrokode må man gøre nogenlunde følgende:

Overfør adressen i **P** til **LAR**, dvs adressen på den næste instruktion, der skal udføres lægges i **LAR** → læs indholdet af lagercellen med adressen i **LAR** ind i **LDR** → send indholdet af **LDR** til **I** → tæl **P** et op.

I mikrokode ser det derfor således ud:

Send indholdet af **P** registret til bussen
Modtag bus-indhold til register **LAR**
Læs lager adressen i **LAR** og læg resultatet i **LDR**
Send indholdet af **LDR** registret til bussen
Modtag bus-indhold til register **I** [1]
Tæl **P** en op

[1] Når I registret modtager data bliver det automatisk splittet op i to: en del til ADR registret og en til I

Samlet set ser de to liniers maskinkode således ud:

Send indholdet af **P** registret til bussen
Modtag bus-indhold til register **LAR**
Læs lager adressen i **LAR** og læg resultatet i **LDR**
Send indholdet af **LDR** registret til bussen
Modtag bus-indhold til register **I**
Tæl **P** en op

Send indholdet af **ADR** registret til bussen
Modtag bus-indhold til register **LAR**
Læs lager adressen i **LAR** og læg resultatet i **LDR**
Send indholdet af **LDR** registret til bussen
Modtag bus-indhold til register **A**

Send indholdet af **P** registret til bussen
Modtag bus-indhold til register **LAR**
Læs lager adressen i **LAR** og læg resultatet i **LDR**
Send indholdet af **LDR** registret til bussen
Modtag bus-indhold til register **I**
Tæl **P** en op

Send indholdet af **A** registret til bussen
Modtag bus-indhold til register **X**
Send indholdet af **ADR** registret til bussen
Modtag bus-indhold til register **LAR**
Læs lager adressen i **LAR** og læg resultatet i **LDR**
Send indholdet af **LDR** registret til bussen
Modtag bus-indhold til register **Y**
ALU Addition

Register A indeholder nu indholdet af adresse 02 og 03 adderet.

3.3 Mikrokode i LINDA

Det er muligt at arbejde med mikrokode på to måder i LINDA. Man kan udføre mikrotrin, det svarer til enkelttrin, blot tages der kun en mikroinstruktion af gangen. Statusbaren nederst i LINDA vinduet viser hvilke mikroinstruktioner maskinen udfører. Derudover er det muligt at ændre i implementeringen af maskinkodeinstruktionerne.

For at ændre i implementeringen skal man blot åbne Maskine.java filen i dk/dyregod/linda kataloget. Nederst i filen findes alle instruktionerne maskinen forstår med en metode pr. instruktion. F.eks. ser `HENT` instruktionen således ud:

```
public void HENT()
{
    sendTilBus(ADR);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(A);
}
```

De tilgængelige mikroinstruktioner er helt analogt med de tidligere definerede:

```
sendTilBus(int register)
modtagFraBus(int register)
laesLager()
skrivLager()
ventForInput()
ALUAdd()
ALUSub()
ALUDiv()
ALUMult()
```

sendTilBus og modtagFraBus tager som argument en int mellem 0 og 7 der angiver hvilket register, der skal arbejdes med. Alle registernumrene har dog også en konstant, som er navnet på registret med store bogstaver. Så hvis man f.eks. vil sende indholdet af register ADR til bussen ville kommandoen skulle se sådan ud (Husk semikolon efter hver linie i java):

```
sendTilBus(ADR);
```

Når man har ændret i Maskine.java filen skal LINDA genkompileres, dette gøres ved at køre compile.bat filen. Herefter kan LINDA programmet startes igen, og nu bruges koden med ændringer.

3.4 Øvelser

Øvelse 3.1

Hvordan vil følgende program se ud i mikrokode? Lav øvelsen uden at bruge LINDA.

Adresse	Indhold	
0:	HENT	06
1:	ADD	07
2:	DIV	08
3:	GEM	05
4:	STOP	
5:	TOM	00
6:	TOM	24
7:	TOM	23
8:	TOM	02

Øvelse 3.2

Åben Maskine.java filen i dk/dyregod/linda/ kataloget og implementer DIV instruktionen. Du finder metoden nederst i filen. Afprøv om instruktionen virker i LINDA ved at køre på compile.bat filen. [2]

Øvelse 3.3

Udvid LINDA maskinens instruktionssæt med endnu en instruktion `KVA` (**Kva**drer) der sætter indholdet af cellen i anden. Instruktionen skal have nummer 13.
HINT: Der skal tilføjes endnu en case sætning til executeNext() metoden, samt laves en ny metode af valgfrit navn til instruktionen.

[2] Hvis noget går galt kan du finde den originale Maskine.java fil i source kataloget

Indeks

Bilag A

Instruktionssæt

En liste over alle de instruktioner LINDA forstår.

Nummer	Navn	Beskrivelse
0	TOM	ingen
1	STOP	stopper afviklingen af programmet
2	HENT	henter indholdet af celle til register A
3	GEM	gemmer indholdet af register A til celle
4	ADD	adderer indhold af celle til A
5	SUB	subtraherer indholdet af celle fra A
6	MULT	multiplicerer register A med indholdet af celle
7	DIV	dividerer register A med indholdet af celle
8	IND	indlæser tal til celle
9	UD	udskriver tal fra celle
10	HOP	hopper til adresse (register p sættes til adresse)
11	HNUL	hopper til adresse hvis A er 0
12	HNEG	hopper til adresse hvis A er negativ

E CD

CD'en indeholder følgende:

- LINDA programmet
- LINDA source
- Java Runtime Enviroment version 1.3.1
- Java Development Kit version 1.3.1
- Netbeans IDE
- Eclipse IDE
- README

BEMÆRK: LINDA programmet kører pt kun under Microsoft Windows. En version til Linux, Solaris og AIX vil være tilgængelig på http://www.dyregod.dk/linda