DR

DANMARKS SKOLERADIO/TV · ELEVHÆFTE

DATA LOGI

7 SKOLERADIO/TV-UDSENDELSER I FORÅRET 1969

Faglig tilrettelæggelse og udarbejdelse af seriens materialer: Chr. Gram, H. B. Hansen og Peter Naur.
Radio-produktion: Jørgen Juulsgaard.
TV-produktion: Palle Mogensen.
Tegninger: Johanne Bojesen.
Lay-out: Poul Jeppesen.
Redaktion: Jørgen Juulsgaard og Palle Mogensen.
Tryk: Th. Laursens Bogtrykkeri, Tønder.

INDHOLD

1. Data, nøglen til det moderne samfund

1.1. Data, en ny betegnelse for gammelkendte ting

I dette hæfte og i de tilhørende udsendelser i radio og TV skal vi beskæftige os med en række hjælpemidler og apparater der alle viser anvendelser af det man nutildags kalder data. Vore eksempler rækker fra ganske dagligdags ting som skrift, vejskilte og kort med huller, over mere indviklede opstillinger med vippelåger der påvirkes af små, tunge brikker, til de store, yderst indviklede apparater der betegnes *datamater*.

Den fælles linie gennem alt hvad vi skal tale om er *data*, hvorved vi forstår alt hvad der kan bruges til at optegne, registrere eller overføre viden og oplysninger. Eksempler på data træffer vi overalt: morsesignaler gennem en ledning; huller i togbilletter; blink i en bils afviserlygte; en færdselsbetjents pegen med sin stav; arkitektens tegning af et hus; et landkort med alle dets linier og signaturer; skriften i dette hæfte; viserne på urskiven; trafiktavlerne med deres bogstaver og signaturer; kridt på en tavle; riller i en grammofonplade; billeder på en fjernsynsskærm; osv. osv.

Som det ses af disse eksempler omfatter data talrige dagligdags fænomener som har været kendt i mange år. Til trods herfor er det først for nylig man er begyndt at tale på denne måde om data. Før omkring 1960 træffer vi næsten ikke denne sprogbrug, og først i 1966 er betegnelsen blevet internationalt anerkendt ved at optræde i en ordbog over tekniske betegnelser fra området Informationsbehandling.

Nutidens situation hvor vi finder at forestillingen om data er naturlig og uundværlig er derimod fremtvunget af en uhørt teknisk og videnskabelig udvikling der tog sin begyndelse omkring 1945.

Denne udvikling er først og fremmest knyttet til konstruktionen af store og indviklede elektroniske apparater, der automatisk og umådelig hurtigt kan behandle data, såkaldte *datamaskiner* eller *datamater*.

1.2. Rødderne til datamaternes revolution

Selv om begivenhederne efter 1945 har rødder langt tilbage i tiden kan vi med rette opfatte det der skete i 1940'rne som et spring i udviklingen. Nøjere beset kan vi endda skelne tre forskellige skridt, der blev taget omtrent på samme tid. Det første var at nogle enkelte pionerer, *Zuse* i Tyskland, *Turing* i England, *Stibitz* og *Aiken* i U.S.A., gik i gang med at konstruere fuldt automatiske regnemaskiner, altså maskiner som uden stadige indgreb fra mennesker er i stand til at gennemføre lange kæder af beregninger. Hidtil havde man kendt små bordregnemaskiner, af den art som stadig ses f. eks. i ban-

A1	DATA	A representation of facts or ideas in a formalised manner capable of being communicated or manipulated by some process.

Fig. 1.1. Faksimile fra ordbogen »IFIP/ICC Vocabulary« der viser definitionen af data

ker, og som kan udføre additioner, subtraktioner, multiplikationer, og divisioner. Men der kræves menneskelig indgriben for hver enkelt regneoperation maskinerne skal udføre. De nye maskiner var langt større, fyldte hele lokaler eller sale, men var til gengæld i stand til automatisk at udføre lange serier af beregninger, hvor det resultat der er nået i en beregning kan bruges i fortsatte beregninger, time efter time, så længe man ønsker. Disse store automatiske regnemaskiner fra omkring 1940 var elektromekaniske, dvs. de var opbygget dels af bevægelige og drejelige dele, som hjul, vippearme, og lignende, dels af elektriske dele, som motorer, kontakter, og elektromagneter. Tillige benyttedes strimler og kort af karton, med huller i bestemte mønstre.

Det andet skridt i udviklingen omkring 1942 var at *Eckert* og *Mauchly* i USA fandt på at erstatte de elektromekaniske dele med rent elektroniske, altså elektriske kredsløb hvor kun elektronerne bevæger sig mens apparatet arbejder. Eckert og Mauchly's maskine hed *Eniac* og var færdig i 1945. Ligesom sine elektromekaniske forgængere var den et monstrum. Den fyldte alle væggene i et stort lokale og indeholdt blandt andet 18.000 elektronrør og en tilsvarende mængde modstande, kondensatorer, og ledninger. Dens store gevinst i forhold til forgængerne var arbejdshastigheden. Mens de tidligere maskiner i ét sekund kunne udføre omkring en enkelt regneproces, f. eks. en addition af to tal, kunne Eniac udføre omkring 1000. Dette var resultatet af at erstatte tunge mekaniske dele med letbevægelige elektroner.

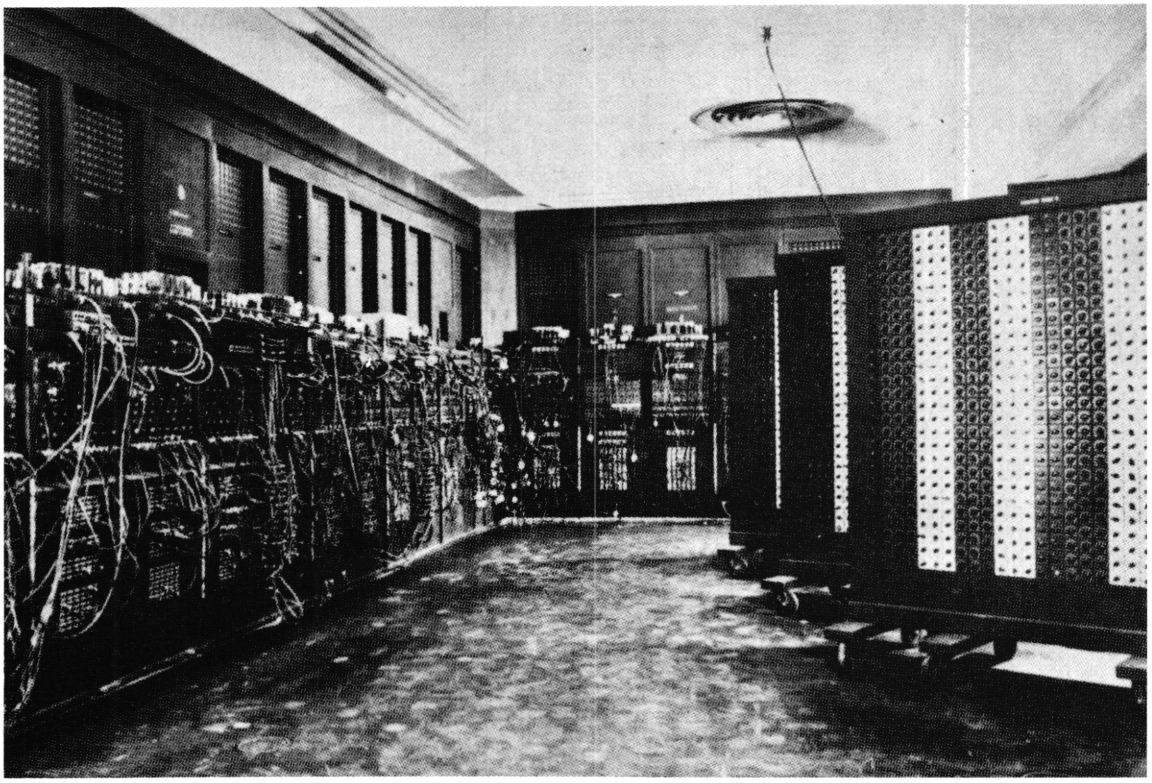
Det tredje skridt der indvarsler den vældige udvikling fra 1945 var at man udtænkte en ny måde at opbygge maskinerne på. Man opdagede at evnen til at regne med tal, som hidtil havde været det væsentligste i bestræbelserne, i virkeligheden ikke var det vigtigste ved de ny maskiner. Langt vigtigere var deres *automatiske virkemåde*, altså deres evne til at fortsætte med at udføre en beregning skridt efter skridt, uden at mennesker behøver at gribe ind. Det springende punkt blev her: hvordan skal vi mennesker give besked til maskinen om hvad det er for en beregning vi ønsker udført? Indtil 1946 benyttede man besværlige metoder hertil. Således måtte man forberede Eniac til at løse en bestemt opgave ved dels at montere et stort antal ledningsforbindelser mellem maskinens dele, dels at indstille mange forskellige knapper rundt om i den.

Den store nydannelse i 1946 var at man gjorde sig klart at det der kræves for at styre en automatisk maskine for at den skal klare en bestemt opgave er forskellige oplysninger, altså en slags data. Disse data kan med stor fordel opbevares af maskinen selv, på ganske samme måde som den opbevarer de tal der bruges i beregningen. Man nåede på den måde frem til at opbygge maskiner omkring et *lager* for oplysninger, som foruden de *talværdier* der behandles kan bruges til at opbevare ordrer for maskinens arbejde, det *program* der styrer maskinen. En afgørende fordel ved denne løsning af problemet at styre maskinerne er at det bliver ganske let at omstille en maskine fra én opgave til en anden. Der kræves blot at man lader maskinen selv læse et andet program ind i sit eget lager, en sag der i moderne datamater kan klares på få sekunder.

Disse tre faktorer: *de gode erfaringer* med store regnemaskiner, udnyttelsen af *elektroniske kredsløb*, og brugen af et *lager* til både at rumme talværdier og programmet der styrer maskinen, udgør tilsammen det spring i udviklingen der fandt sted i løbet af 1940'rne. De nye ideer slog igennem i praksis i 1949, da den første datamat der kombinerer alle disse principper, maskinen *Edsac*, blev bragt til at fungere i England. Som vidnesbyrd om de nye ideers bærekraft kan det anføres at Edsac med sine 3000 elektronrør var både mere ydedygtig og umådelig meget bekvemmere i brug end Eniac med sine 18.000 elektronrør.

1.3. Datamaterne breder sig

Den følgende udvikling blev præget af at man opdagede at de nye maskiner ikke kun kan bru-

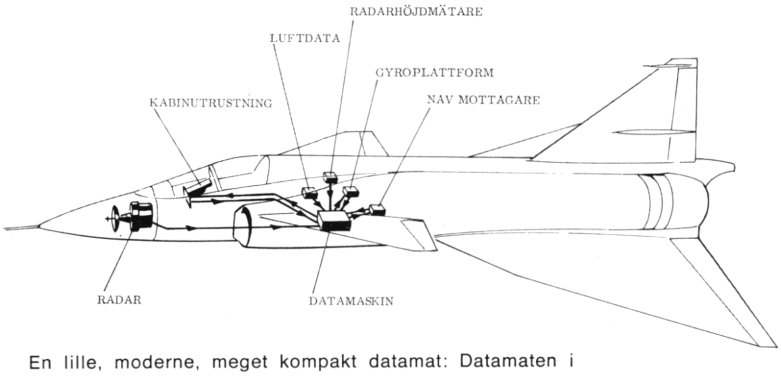


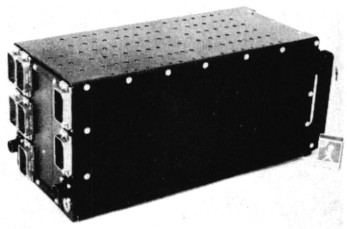
ENIAC, den første næsten fuldstændig elektroniske datamat, stod færdig i 1945. Den blev planlagt og konstrueret af de to ingeniører **J. P. Eckert** og **J. W. Mauchly** ved Pennsylvania Universitet. Datalageret kunne kun rumme 20 tal hver med højst 10 cifre (titals-cifre), men dette lager skulle ikke rumme programmer. Programmeringen af ENIAC foregik ved at man indstillede en lang række kontakter og trak en masse ledninger. Til gengæld for den meget besværlige programmering regnede ENIAC hurtigt når den først var programmeret: Addition af to tal tog kun $^{2}/_{10000}$ sekund, og multiplikation af to tal tog $^{30}/_{10000}$ sekund. Skønt den indeholdt henved 18.000 elektronrør og 1500 relæer var den forbløffende driftssikker takket være et første klasses konstruktionsarbejde.

ges til beregninger, men tværtimod egner sig fortrinligt til at behandle *tegn* af enhver art, både cifre, bogstaver, og andre slags, som komma, semikolon, plustegn, parenteser, osv. altså data i videre forstand. Denne slags anvendelser er blevet stadig mere fremtrædende og har gradvist ført til forståelse af at de nye maskiner rettelig bør opfattes som databehandlingsmaskiner, og man er derfor gået over til at betegne dem som *datamaskiner* eller *datamater*. Denne nye forståelse af maskinernes mest grundlæggende egenskaber er gået hånd i hånd med et nyt syn på alt det vi nu kalder data, altså skrift, signaler, osv. Vi har indset at det er hensigtsmæssigt at tale om et særligt fag, *datalogien*, læren om data. Datalogien omfatter enhver form for data, uanset om den er knyttet til datamater eller ej. En vigtig del af datalogien handler om *datamatik*, dvs. behandling af data med automatiske hjælpemidler, især med datamater.

De nye ideer fra 1940'rne var umådeligt frugtbare. I løbet af få år blev de taget op mange forskellige steder, både ved universiteter og i industriforetagender. I løbet af 1950'erne blev der bygget hundreder af datamater, og i 1967 var tallet nået op over 50.000. Samtidig blev de stadig forbedrede. Således er vor tids mest ydedygtige datamater omkring 1000 gange så hurtige som Eniac. De kan altså udføre omkring 1 million additioner pr. sekund.

Den enorme vækst i datamaternes antal og





En lille, moderne, meget kompakt datamat: Datamaten i det svenske jagerfly »Viggen« består af i alt 5 apparatkasser som den der er vist til højre på figuren. Hver kasse er tæt pakket med elektroniske dele og kredsløb og fylder ikke mere end en lille skrivebordsskuffe. Datalageret kan rumme ca. 8000 8-cifrede tal og regnehastigheden er 50–100 gange så stor som for ENIAC. Datamaten skifter automatisk mellem flere programmer der er anbragt i datalageret, og disse skift sker så hurtigt at det for piloten virker som om datamaten udfører følgende funktioner samtidigt: (1) Løbende navigationsberegninger, (2) sigte- og skydeberegninger, (3) løbende registrering af brændselsbeholdningen og (4) løbende kontrol af at datamaten selv og de tilsluttede måleapparater fungerer korrekt.

ydeevne er blevet stærkt fremskyndet af at man stadig har fundet nye nyttige anvendelser for dem. De første datamater var mest beskæftigede med videnskabelige opgaver, ikke mindst beregninger af hvordan atomreaktorer opfører sig. I dag er vi for længst nået til at langt det meste arbejde der udføres af datamater ikke har meget med videnskab at gøre. Inden for erhvervsvirksomhederne bruges de til lønningsberegninger, til kundebogholderi, til at planlægge produktionen, til at styre værkstedsmaskiner, og meget mere. Staten bruger dem til skatteberegninger, til at planlægge nye veje, til at holde styr på studenterne ved universiteterne, og meget andet.

I stigende grad benyttes datamater som hjælpemiddel dér hvor der skal træffes vigtige beslutninger. Skal der bygges en bro over Øresund, og i så fald hvor? For at besvare sådan et spørgsmål må vi finde ud af hvad følgerne ville være hvis vi byggede den. Hvor skulle pengene tages fra? Hvilken indflydelse ville det have på andre ting der behøver penge, f. eks. bygning af skoler? Hvordan ville trafikken blive hvis der var en bro? Hvilken indflydelse ville en bro have på hvor folk bor? Ville en masse mennesker flytte til Skåne fra Sjælland, eller omvendt? Disse og en række andre spørgsmål må besvares på en eller anden måde før vi kan beslutte os til at bygge en bro. Imidlertid er de meget vanskelige at besvare, og det vil kræve omfattende undersøgelser over hvordan en bro vil påvirke hele samfundsmønsteret omkring Øresund at finde blot nogenlunde gode svar. Det er ved sådanne opgaver at datamaterne bliver stadig mere afgørende. Kun med deres hjælp er vi i stand til at holde styr på de enorme mængder data der kræves for at beskrive et samfund med alle dets borgere og aktiviteter.

Men en datamat er ikke en trylleformular eller Aladdins lampe. Den gør kun hvad mennesker har foreskrevet at den skal gøre. Selv om datamaten kan være til enorm hjælp næsten overalt i samfundet, så afhænger resultatet af dens arbejde helt afgørende af de folk der tilrettelægger opgaven for den.

1.4. Datamaterne i det demokratiske samfund

I et demokrati er det afgørende at alle borgere har mulighed for at forstå de afgørelser der træffes. Kun hvis denne forståelse er til stede

er det rimeligt at tale om en medindflydelse. Når et magtfuldt hjælpemiddel som datamaterne i stigende grad bliver brugt ved de store afgørelser, da er det vigtigt for demokratiet at alle borgere får en forståelse af hvad datamater kan og hvad de *ikke* kan. Især er det vigtigt at borgerne forstår hvor stor indflydelse den person der tilrettelægger opgaven for datamaten har på resultatet af datamatens arbejde.

Disse overvejelser ligger til grund for udformningen af denne skoleradio/TV-serie. Vi tager vort udgangspunkt i selve forestillingen om data. Denne kan udtrykkes ved følgende definition, der på grund af forestillingens altomfattende natur ikke kan undgå at være vanskeligt forståelig:

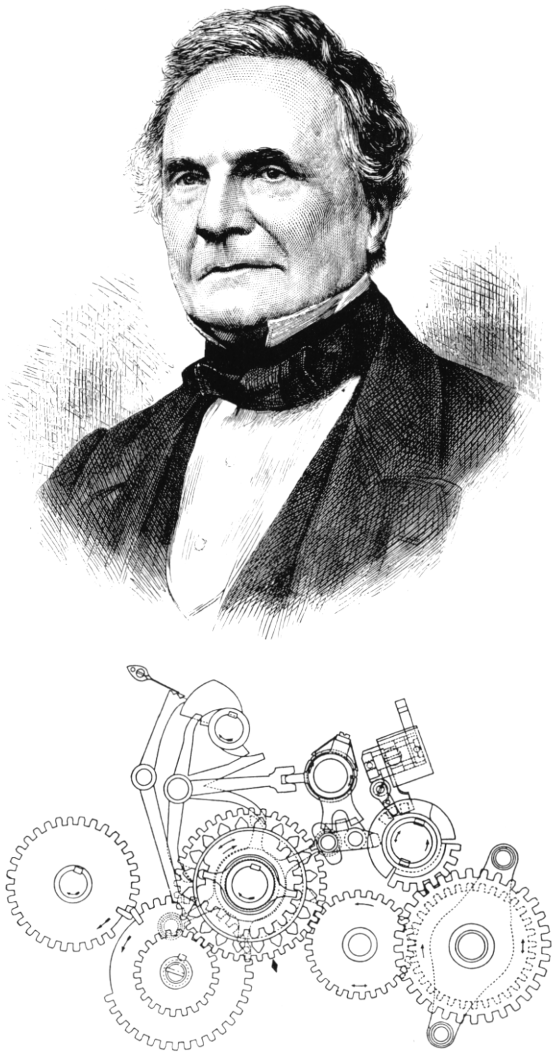
> Data: En formaliseret repræsentation af kendsgerninger eller forestillinger på en sådan form, at den kan kommunikeres eller omformes ved en eller anden proces.

Man må ikke lade sig afskrække af denne korte form. Faktisk kan den første del af serien betragtes som en forklaring af hvad denne definition siger. Vi skal vise en række eksempler på *data*, på de *kendsgerninger* eller *forestillinger* de *repræsenterer*, og på de *processer* de kan udsættes for.

Vi skal særlig interessere os for sådanne data som egner sig til at behandles med mekaniske eller elektriske midler, med ringe menneskelig indsats. Herigennem vil vi gradvis opbygge grundlaget for at forstå den vidtdrevne automatisering der træffes i datamaterne.

Med dette grundlag skal vi se nøjere på hvordan en opgave må omformes fra en form som vi alle umiddelbart kan forstå til den form der kræves ved løsning med en datamat. Dette vil understrege at selv når ydedygtige datamater står til rådighed ligger der en meget betydningsfuld indsats hos de folk der lægger opgaverne til rette.

Skoleradio/TV-serien slutter med nogle oplysninger om datamaternes fremmarch og de erhvervsmæssige muligheder der følger deraf.



Den engelske matematikprofessor **Charles Babbage** (1792–1871) var den første der indså muligheden af at konstruere en automatisk maskine til databehandling. Hans »analytiske maskine« benytter mange af de datalogiske principper, der kendes fra vore dages datamater (man kan nævne programstyring samt trykning af data der muliggør viderebehandling ad automatisk vej). Babbage måtte imidlertid basere sine konstruktioner på finmekanik, da det var den eneste teknik man beherskede nogenlunde på hans tid. Dette viste sig at rejse uoverstigelige vanskeligheder. Det lykkedes ham aldrig at konstruere en tandhjulsudveksling med det nødvendige antal tandhjul og vægtstænger som ikke låste sig selv fast når den blev sat i gang. Efter 25 års ihærdigt arbejde og tabet af sin formue måtte Babbage opgive at se sin drøm realiseret. På billedet ses en detalje af en af de tusinder af arbejdstegninger han udarbejdede.

2. Simple mekaniske dataprocesser

2.1. Data som kan afføles mekanisk

Den første betingelse man må stille til en mekanisme, der skal være brugelig til automatisk databehandling, er, at mekanismen kan *afføle* de data der skal behandles. Når databehandlingen sker med menneskelig arbejdskraft sker denne afføling ved at man *læser* de data, der er tale om, altså ved hjælp af synssansen, eller ved at man *hører* dataene med høresansen. Disse afføling smetoder er så indviklede, at man endnu ikke har kunnet efterligne dem rent teknisk, og vi må derfor ty til mere primitive metoder.

Et eksempel på en mekanisme til afføling af data har man i telefonens nummerskive(fig.2.1). Ved at stikke fingeren i et af hullerne udvælger man det bogstav eller det ciffer man ønsker. Når man herefter drejer skiven så langt at fingeren støder imod »stopklodsen« får man sat mekanismen i en tilstand der muliggør mekanisk afføling; informationen (oplysningen) om hvilket hul der er valgt er så at sige gemt i mekanismen i form af hvor langt skiven er drejet. Når man slipper skiven sættes selve *afføling sprocessen* i gang. Skiven er påvirket af en fjeder der bringer den tilbage til sin udgangsstilling, og under tilbageløbet frembringes ved hjælp af en kontakt nogle elektriske impulser – jo flere impulser desto længere skiven skal dreje sig. Mekanismen kan på denne måde *tælle* sig frem til hvilket hul man har valgt.

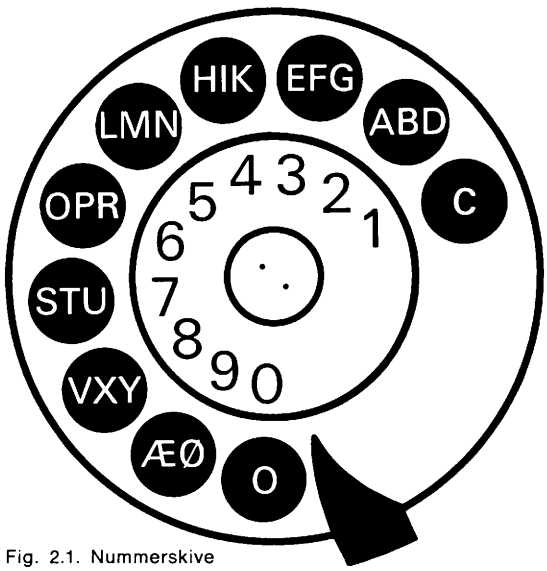


Fig. 2.1. Nummerskive

Nummerskiven er et eksempel på en mekanisme der omdanner data til en form der er yderst velegnet til viderebehandling ad automatisk vej, nemlig elektriske impulser der bevæger sig langs en ledning.

Vi skal mest beskæftige os med et andet afføling sprincip som er mere velegnet når en del af databehandlingen udføres manuelt (»ved håndkraft«). De fleste ved sikkert at der er noget der hedder *blindeskrift*, og at det er en form for skrifttegn som blinde mennesker kan *føle* med fingerspidserne. Blindeskriften fremtræder som små *forhøjninger* på tekstsiden (se fig. 2.2). Det er antallet af forhøjninger og disses indbyrdes placering der over for den blinde angiver hvilke bogstaver han har at gøre med. Et sådant princip ville kunne anvendes til mekanisk afføling, men hvis forhøjningerne erstattes af *huller*, igennem hvilke man f. eks. kan stikke en *følepind* der slutter en elektrisk kontakt, har man en konstruktion der er langt mere tilfredsstillende i mekanisk henseende.

Ideen med huller bliver brugt inden for databehandling på mangfoldige måder. Vi skal se lidt nærmere på de såkaldte *kanthulkort*, der kan se ud som vist på fig. 2.3. Et kanthulkort er

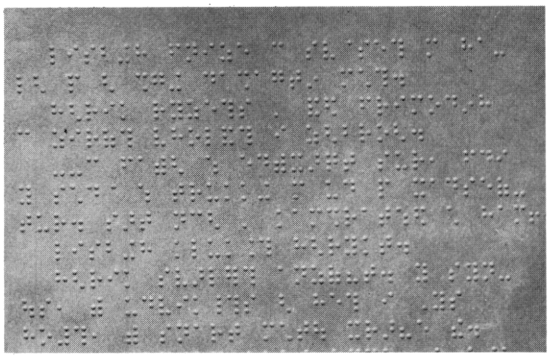


Fig. 2.2. Eksempel på blindeskrift

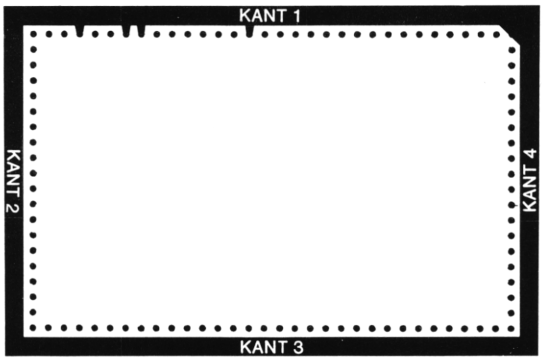
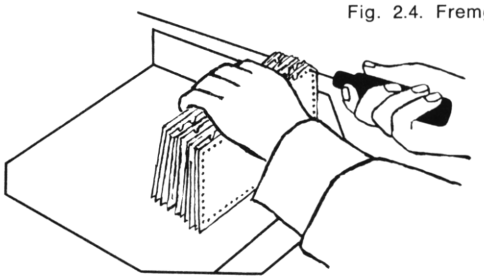


Fig. 2.3. Et kanthulkort

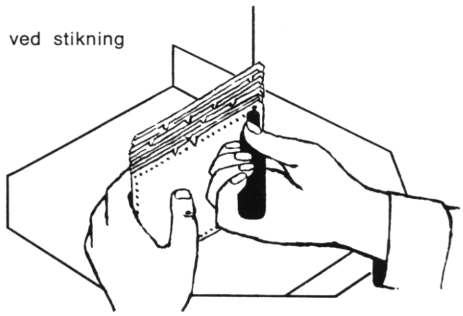
fremstillet af karton og er forsynet med huller langs alle fire kanter. Det ene hjørne er afskåret så man kan se om en stabel kort er placeret rigtigt i forhold til hinanden. Når kortet er retvendt med det afskårne hjørne øverst til højre, som på fig. 2.3, nummereres kanterne som vist på figuren.

Til at begynde med er der altså huller hele vejen rundt langs kanten. Man kan nu klippe en U-formet udskæring ind til nogle af hullerne som vist på fig. 2.3, kant 1. En sådan udskæring vil vi kalde en *markering*. Lad os forestille os at vi har en hel stak kanthulkort med markeringer i. Vi ved intet om hvordan de enkelte kort er markeret. Kortene lægges sammen, idet vi ved hjælp af de afskårne hjørner sørger for at alle kortene ligger orienteret på samme måde. Derefter rejser vi kortbunken til lodret stil-

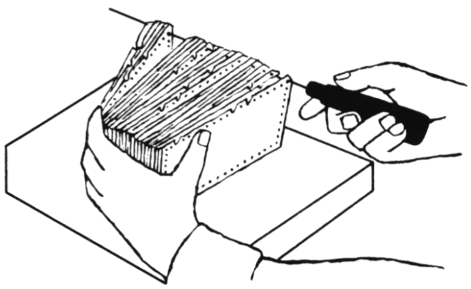
Fig. 2.4. Fremgangsmåden ved stikning



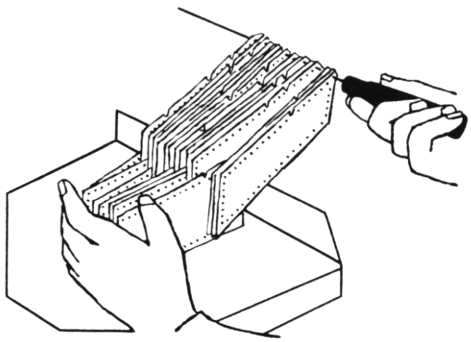
a. Man holder kortbunken, der støtter på bordet, løst med venstre hånd, mens strikkepinden holdes i højre hånd. Den kant, der skal stikkes, vender opad.



b. Når strikkepinden er stukket gennem det valgte hul, drejes den i vandret plan, så kortene forskydes i forhold til hinanden.



c. Kortene fastholdes med venstre hånd, mens strikkepinden i højre hånd drejes tilbage. Herved fremkommer en kortvifte og de enkelte kort kan komme fri af hinanden.



d. Strikkepinden løftes og rystes let. Herved falder de markerede kort ned på bordet. Man kan godt holde løst på kortbunken med venstre hånd, så de markerede kort ikke vælter.

ling, og stikker en strikkepind ind i et af hullerne. *Når vi løfter strikkepinden vil de kort der har en markering i det valgte hul blive liggende på bordet,* mens de umarkerede kort vil hænge fast på strikkepinden. På denne måde kan man helt mekanisk afføle hvilke kort der har markeringer i bestemte huller. I praksis foregår stikkeprocessen lettest ved en fremgangsmåde som vist på fig. 2.4. For at få de markerede kort til at slippe strikkepinden let, må selve markeringen være udført omhyggeligt. Der findes særlige tænger til dette formål, men med lidt omhu kan en almindelig papirsaks anvendes.

Vi ser at der er en nøje overensstemmelse mellem en nummerskive, en side med blindeskrift, og et kanthulkort. Alle er de eksempler på mekanismer der kan indeholde data på en måde der gør det muligt at aflæse disse data igen. Blot forekommer data på forskellig måde i de tre mekanismer. Man siger at data kan *repræsenteres* i mekanismerne, og at der er forskel i *datarepræsentationen.*

Når man skal bruge kanthulkort til et bestemt formål sker det ved at man tillægger markeringerne en eller anden betydning. Lad os f. eks. antage at man ønsker at bruge kanthulkort i forbindelse med en registrering af motorkøretøjer. Der kan være mange ting der har interesse i denne forbindelse; vi kan nævne motorkøretøjets art (personbil, lastbil, varevogn, motorcykel etc.), lasteevne, chassis nr., indregistreringsnummer, motortype (totakts eller firetakts, benzin- eller dieselmotor) osv., osv. På fig. 2.5 er vist, hvordan man kan tænke sig motorkøretøjets *art* registreret. Man har til dette formål anvendt de fire huller på kant 1 der ligger nærmest det afskårne hjørne. Det er naturligvis ligegyldigt hvilke huller man anvender når man blot anvender de samme huller på alle kortene. Hvis registreringen sker i forbindelse med vognparken for en industrivirksomhed er det måske slet ikke nødvendigt at skelne mellem så mange arter som på fig. 2.5. Måske ønsker man kun at skelne mellem lastvogne, varevogne og andre motorkøretøjer. Man kan da nøjes med at reservere tre huller til dette formål. Man kan endog nøjes med to huller, f. eks. lastvogne og varevogne. Når man skal finde kortene for »andre motorkøretøjer« gør man da følgende:

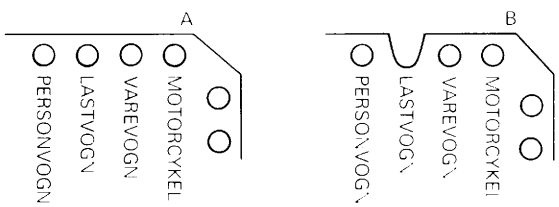


Fig. 2.5. Registrering af motorkøretøjets art
a. Bestemte huller har bestemt betydning
b. Markering af en lastvogn

1. Stik strikkepinden i »lastvogne« og gå videre med de kort der bliver hængende.
2. Stik strikkepinden i »varevogne«. De kort, der bliver hængende, er da »andre motorkøretøjer«.

Det bliver altså lidt besværligt at finde de umarkerede kort, men til gengæld sparer man et hul, som man måske har hårdt brug for til et andet formål. Da man er frit stillet med hensyn til, hvad hullerne skal betyde, er det naturligvis det klogeste at lade dem betyde det man oftest søger (hvis man næsten aldrig er interesseret i varevogne skulle hullerne i eksemplet have heddet »lastvogne« og »andre motorkøretøjer«).

De overvejelser vi her har gjort os om hullernes betydning, er ikke specielle for kanthulkortet, men gælder i almindelighed. De kan sammenfattes på følgende måde:

> Data kan i en given sammenhæng altid repræsenteres på flere forskellige måder. Valget af repræsentationsform må træffes ud fra kendskabet til den brug man vil gøre af dataene, idet kapacitetsomkostningerne (ved kanthulkort: antallet af huller) afvejes mod omkostningerne ved selve databehandlingen (ved kanthulkort: antallet af stik).

Man kunne nu spørge hvilken glæde man har af at udvælge bestemte kort af en kortstak

Ryg: Slå 120 m op, strik 4 cm i mønster 1, fortsæt i mønster 2. Når arb måler 38 cm fra beg, lukkes 3 m af til raglanærmg, derefter tages 1 m ind i hver side på hver 2. p 39 g (= i beg af p: kantm, 1r, 2r sm, i slutn af p: 2r sm dr, 1r, kantm). Når der er 42 m tilbage på p, lukkes de midterste 18 m af til halsudskæring, derefter lukkes i halssiden 5 m 1 g og 4 m 1 g.

Forstykke: Slå 120 m op, strik 4 cm i mønster 1, tag 4 m jævnt ud på den følg p og strik i følg inddeling: kantm, 54 m i mønster 1, 14 m i mønster 3, 54 m i mønster 2, kantm. Når arb måler 16 cm, strikkes 1r dr i lænken mellem de 2 vrangm på midten. På den 6. p efter denne udtagne m tages 2r sm før snoningsstriben, på hver side af

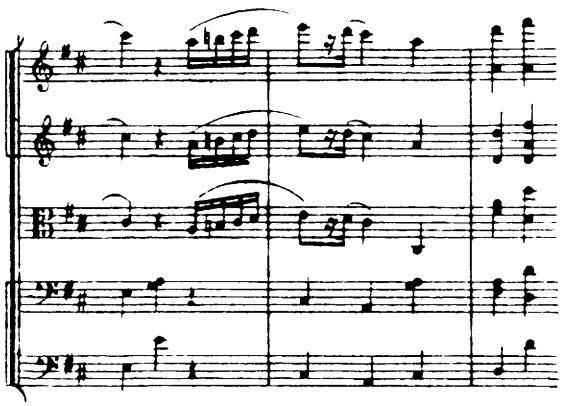


Fig. 2.6. Brudstykker af strikkeopskrift og partitur

på denne måde. Svaret er at man ønsker at se nogle af de andre ting, der står på de udvalgte kort. Man kan f. eks. lade hullerne betyde noget om bøger i et bibliotek: spændende bøger, drengebøger, kriminalromaner, slægtsromaner, bøger om havet, bøger om kærlighed, osv. så langt fantasien eller antallet af huller rækker. Hver bog får da sit kort med markeringer i de huller der karakteriserer netop denne bog. På det åbne felt på midten af kortet skriver man bogens forfatter og titel, samt evt. et resumé. Man kan da let finde de bøger der handler om noget man interesserer sig for. Ønsker man at læse en spændende bog om søfolk, tager man kortstakken og stikker først i »spændende bøger«. De kort, der falder fra, stikker man derefter i »bøger om havet«, og man kan da udvælge sig en bog blandt titlerne på de kort der falder fra efter dette stik.

Som vi har set af eksemplerne indtil nu kommer man ofte ud for at skulle udføre sammensatte stikkeoperationer med mere end ét stik. Man gør da klogt i at nedskrive fremgangsmåden inden man begynder selve udvælgelsesprocessen. Denne nedskrivning lettes i høj grad hvis man benytter en eller anden forkortet skrivemåde, et *sprog* der er velegnet til at udtrykke de operationer man kan udføre med kanthulkortene.

Denne fremgangsmåde – at indføre et specielt sprog – er velkendt overalt hvor man har det problem at skulle beskrive en fremgangsmåde til frembringelse af et bestemt resultat. Man kan blot tænke på en *strikkeopskrift* eller et *partitur*, hvor de anvendte sprog ligger meget langt fra dagligdagens sprog. Også kogebogslitteratur eller vejledninger til udfyldelse af formularer er eksempler på sprogbrug der er specielt rettet mod beskrivelse af handlingsforløb eller processer inden for disse områder.

Et sprog der er lavet til beskrivelse af databehandlingsprocesser kaldes et *programmeringssprog*. De sætninger man kan skrive i et givet programmeringssprog kaldes *ordrer*. En samling af ordrer, der udtrykker løsningen til en databehandlingsopgave, kaldes et *program*. Et program svarer altså til en novelle eller en roman i det dagligdags sprog. Det at skrive et program kaldes *programmering*.

Hvis man tænker lidt over sagen vil man kunne se at man for kanthulkorts vedkommende har brug for følgende:

1. at kalde en bestemt bunke ved navn.
2. at kalde et bestemt hul i et kort ved navn.
3. at dele en navngiven bunke i to ved et stik i et navngivet hul.
4. at lægge to bunker sammen til een.

Vi vil for simpelheds skyld vedtage at *nummerere* hullerne i et kort, f. eks. venstre om begyndende med kant 1 nærmest det afskårne hjørne. Lad os herefter antage at vi skal finde de kort i en kortbunke, der har markering i hul 7 eller hul 13, eller eventuelt har markering i begge disse huller. Vi kan da beskrive fremgangsmåden i et programmeringssprog, og dette kan f. eks. se således ud:

Trin	*Ordrer*	*Forklaring*
1	ALLE = bord	De kort der ligger på bordet inden vi starter, kaldes for ALLE.
2	stik 7 i ALLE	Bunken ALLE deles i to ved et stik i hul nr. 7.
3	BUNKE 1 = bord	Den del der forbliver på bordet kaldes BUNKE 1. Disse kort har markering i hul 7 (og evt. i hul 13).
4	BUNKE 2 = pind	Den del af ALLE der hænger på pinden kaldes BUNKE 2.
5	stik 13 i BUNKE 2	BUNKE 2 deles ved et stik i hul nr. 13.
6	BUNKE 3 = bord	De kort der ved denne deling forblev på bordet kaldes BUNKE 3. Disse kort har markering i hul nr. 13.
7	RESULTAT = BUNKE 1 + BUNKE 3	BUNKE 1 og BUNKE 3 sammenlægges til én bunke der kaldes RESULTAT. Denne bunke vil indeholde de søgte kort.

De syv ordrer, skrevet i denne rækkefølge, udgør et *program* til løsning af den stillede opgave. Det programmeringssprog vi har indført består af følgende bestanddele:

1. Reserverede ord (ord med små bogstaver)

1.1. stik. En betegnelse for selve stikkeoperationen.
1.2. bord. En betegnelse for de kort der efter et stik forbliver på bordet.
1.3. pind. En betegnelse for de kort der efter et stik bliver hængende på pinden.

2. Navne (ord med store bogstaver eller cifre)

Der findes to slags navne, nemlig navne på huller i et kort og navne på kortstakke. Disse to slags navne kan kendes fra hinanden ved at hullernes navne består af lutter cifre, mens navne på kortstakke skal indeholde bogstaver.

3. Tegn

3.1. = (lighedstegn). Dette tegn bruges ved navngivning af kortstakke. Den stak der er angivet til højre for lighedstegnet skal kaldes det der står til venstre for lighedstegnet. Dette betyder at stakken skal lægges til side idet man husker dens navn. Dette navn vil senere kunne forekomme på højre side af et lighedstegn eller som sidste ord i en sætning der begynder med »stik«.

3.2. + (plustegn). De stakke der er nævnt til venstre og til højre for plustegnet lægges sammen så de danner én stak.

Dette er naturligvis ikke den eneste måde at lave programmeringssproget på; måske er det ikke engang den bedste måde. Det væsentlige er imidlertid ikke sprogets form, men derimod at vi nu kan indse, at når man først har nedskrevet et program til løsning af en given opgave, er den resterende del af arbejdet begrænset til noget rent mekanisk, nemlig til slavisk at udføre de ordrer der står i programmet.

Denne erkendelse viser os noget meget væsentligt, nemlig at *det i princippet må være muligt at konstruere en maskine der kan adlyde et program*. Eksempler på sådanne maskiner vil blive gennemgået i den 4. udsendelse.

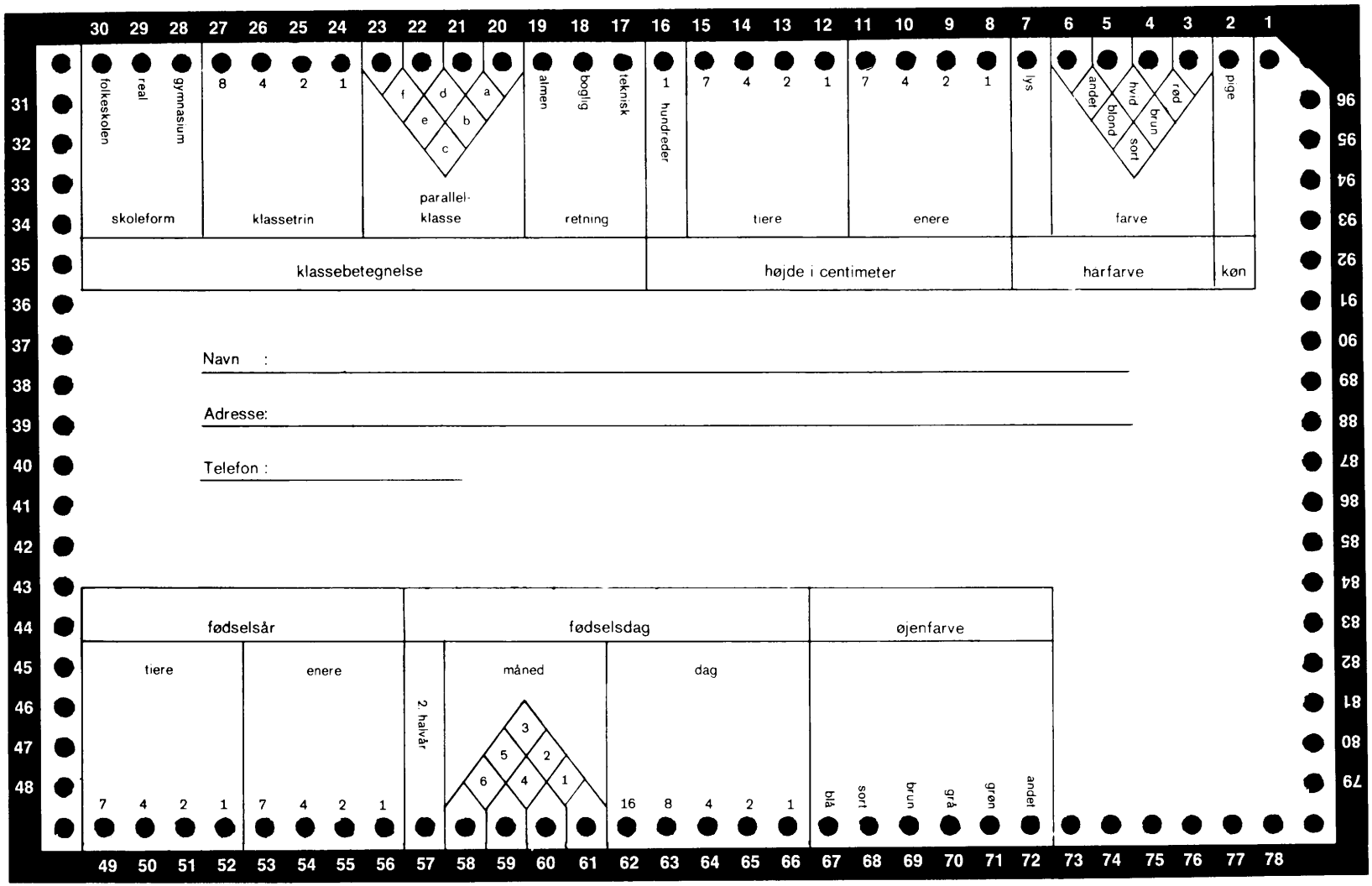


Fig. 2.7. Kanthulkort til elevregistreringsformål

Klasseøvelse

Inden vi forlader kanthulkortene vil vi beskrive en øvelse der går ud på at danne et register med oplysninger om skoleelever. Formålet med registreringen er at finde frem til elever med bestemte personlige egenskaber såsom alder, højde, hårfarve m. v. Det er meningen at hver enkelt elev selv skal angive hvilke markeringer der skal foretages for hans eller hendes eget vedkommende.

Til øvelsen anvendes et kort som vist på fig. 2.7. Vi vil ikke direkte beskæftige os med den vigtige fase i problemløsningen, der består i at bestemme sig til *hvilke* data der skal registreres, samt *hvordan* disse data skal repræsenteres på kortet. Baggrunden for den valgte opbygning af kortet vil dog til dels fremgå af den følgende orienterende gennemgang af kortets felter.

Midt på kortet skal elevens navn, adresse og telefonnummer skrives. Disse oplysninger er ikke markeret på kortet, og man kan derfor ikke ad mekanisk vej opsøge bestemte personer. Derimod kan man opsøge personer med bestemte egenskaber, nemlig de egenskaber der findes markeret langs kanterne. Hvilke egenskaber det drejer sig om fremgår af teksten på selve kortet (se fig. 2.7).

På hulkortet er kun ét hul nødvendigt til registrering af om en elev er *dreng* eller *pige* da der kun er to muligheder. Vi vælger at lade markering betyde »pige« og ingen markering betyde »dreng«.

Til *øjenfarven* er afsat 6 huller (hul nr. 67 til 72). Registreringen foregår ved at man markerer én og kun én af disse muligheder. Man *kunne* have sparet hullet »andet«. Elever med andre øjenfarver end de nævnte ville da ingen markering få, og det ville kræve fem stik at finde dem.

På fig. 2.8 ses feltet til registrering af *hårfarven* i forstørret udgave. Længst til venstre er der et hul til registrering af »lys« eller »mørk« (lyshårede markeres). Selve farven registreres i hullerne 3 til 6 idet man benytter skemaet under hullerne som rettesnor. Hvert felt i dette skema

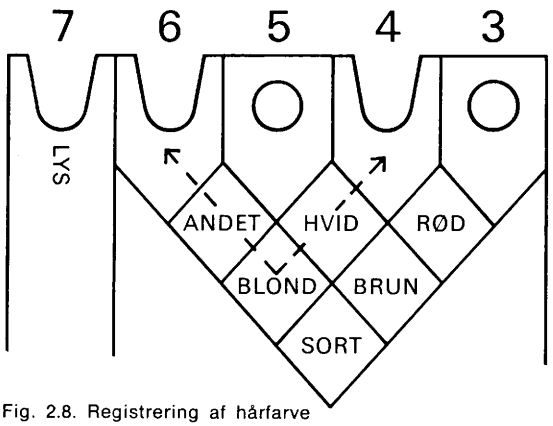


Fig. 2.8. Registrering af hårfarve

udpeger to af de fire huller, når man sigter gennem de af feltets sider, der vender mod hullerne. F. eks. ses, at feltet »blond« udpeger hullerne 4 og 6. Markeringen på fig. 2.8 betyder altså »lysblond«. På denne måde har man med 5 huller dækket i alt 12 forskellige hårfarver, hvoraf dog nogle er tvivlsomme i praksis (f. eks. »lyssort«). Den bedre huludnyttelse er imidlertid sket på bekostning af besværet med at finde personer med en bestemt hårfarve, idet denne operation kræver mindst 2 stik.

Når man skal registrere *tal* i et kanthulkort kunne man gøre det ved at afsætte 10 huller til hvert ciffer og tillægge hullerne værdierne 0 til 9. Det er ganske klart at denne metode er for pladskrævende hvis man har at gøre med store tal. Man må derfor forsøge at klare sig med færre huller pr. ciffer. En måde er at afsætte 4 huller til hvert ciffer, og tillægge hullerne værdierne 1, 2, 4 og 7.

På fig. 2.9 ses hvordan cifrene 0 til 9 markeres efter dette system. Man sørger simpelt hen for at summen af talværdierne for de markerede huller giver det ønskede ciffer. Fordelen ved at vælge talværdierne 1, 2, 4 og 7 er, at der højst bliver to markeringer pr. ciffer.

Fødselsåret markeres på denne måde, idet man dog kan nøjes med de to sidste cifre i årstallet (alle elever må være født i det 20. århundrede). Registrering af *legemshøjden i cm* følger også denne metode, men da der sandsynligvis ikke findes elever på over 2 meter, kan vi nøjes med et enkelt hul til hundrederne.

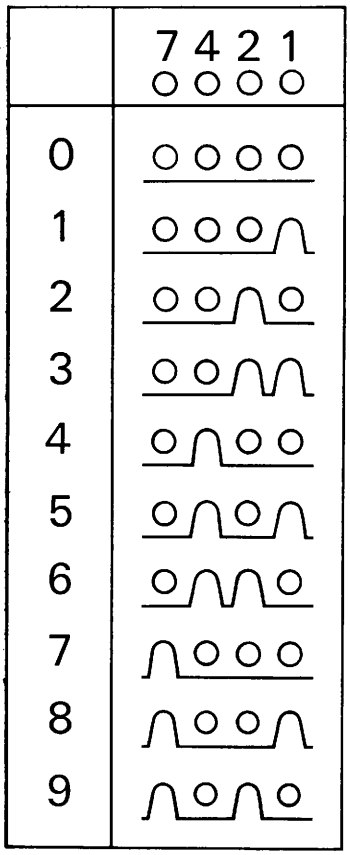
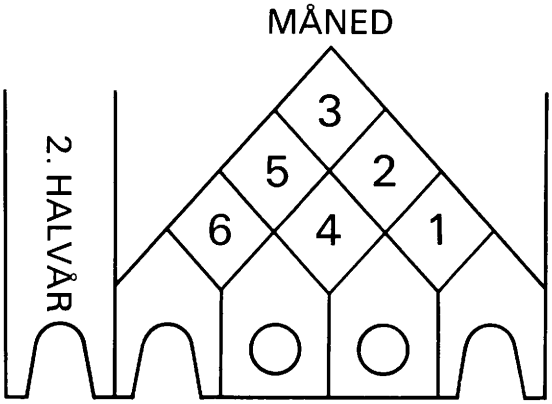


Fig. 2.9. Cifferkoder

På det i øvelsen anvendte hulkort findes også en anden talrepræsentation, nemlig i feltet for *fødselsdatoen* og i feltet for *klassetrinnet*. Ved denne repræsentationsform, der kaldes *binær repræsentation*, tillægges hullerne talværdierne 1, 2, 4, 8, 16, 32, ... (hvert tal er det dobbelte af det forrige tal). Summen af markeringerne skal give det ønskede tal. Det største tal man kan registrere er naturligvis det, der har markeringer i alle hullerne. For datoens vedkommende bliver det 16 + 8 + 4 + 2 + 1 = 31, hvilket netop passer til formålet.

I de øvrige felter på kortet indføres ingen nye repræsentationsformer. Vi vil derfor blot med et par eksempler vise hvordan registreringen skal foregå.

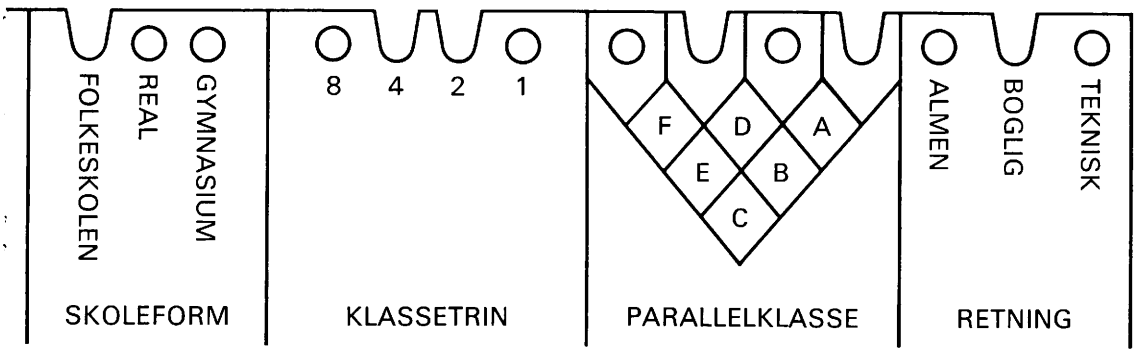
September (2. halvår, 3. måned) registreres således:



6b, boglig retning, vil se ud som vist nederst på siden.

Hvis man har at gøre med en gymnasieklasse, kan man tillægge de tre huller i feltet »retning« betydningerne »matematisk«, »nysproglig« og »klassisksproglig«. Findes der mere end tre retninger, kan man måske klare sig ved at ombytte betydningen af felterne »parallelklasse« og »retning«.

Når alle elever er registreret på denne måde kan man begynde at arbejde med elevregistret. Lad os f. eks. tænke os at alle skolens elever



fra 1. klasse og opad er blevet registreret, og at vi får til opgave at finde alle rødhårede elever med fødselsdag før 1. januar 1960.

Det er let nok at finde alle de rødhårede – det kræver blot et stik i hul 3 og hul 4 (se fig. 2.8). Vi kan helt præcist beskrive fremgangsmåden ved at skrive et *program* i det programmeringssprog vi tidligere har indført:

Trin	*Ordrer*	*Forklaring*
1	ALLE = bord	Alle kort på bordet
2	stik 3 i ALLE	med markering i hul nr. 3
3	VIDERE = bord	går videre
4	stik 4 i VIDERE	Markering i hul nr. 4 blandt disse
5	RØDHÅREDE = bord	udpeger de rødhårede.

Hvis der ikke er nogle kort på bordet efter trin 5 i dette program, er vi færdige, for så findes der ingen rødhårede i skolen. Hvis der derimod er mange rødhårede, står vi nu over for det problem at dele bunken RØDHÅREDE i to: dem med fødselsår 1960 eller senere, samt resten. Vi må altså interessere os for feltet »fødselsår«, men vi ser, at det i virkeligheden kun er *tierne* der har interesse. De rødhårede elever hvis fødselsår begynder med et af cifrene 1, 2, 3, 4 eller 5 er svar på opgaven, mens begyndelsesciffer 6, 7, 8 eller 9 må forkastes. Nu er det naturligvis klart at visse af disse cifre er umulige i praksis, men hvis vi ser bort fra det, får vi et program der kan bruges uændret helt frem til år 2000.

Ved at betragte cifferkoderne på fig. 2.9 og mærke sig egenskaberne ved markeringen af de forskellige cifre, kan vi udlede følgende regler:

1. Alle kort med markering i 7-hullet må forkastes.
2. De kort der har markering i 4-hullet må forkastes hvis de også har markering i 2-hullet.
3. De kort der ikke forkastes ved anvendelse af reglerne 1 og 2 er svar på opgaven.

Herefter kan vi fortsætte programmet.

Trin	*Ordrer*	*Forklaring*
6	stik 49 i RØDHÅREDE	Rødhårede uden markering i 7-hullet
7	VIDERE = pind	går videre.
8	DUER IKKE = bord	Resten lægges til side.
9	stik 50 i VIDERE	Ingen markering i 4-hullet
10	LØSN 1 = pind	er de første løsninger.
11	VIDERE = bord	Resten går videre.
12	stik 51 i VIDERE	Ingen markering i 2-hullet
13	LØSN 2 = pind	er også løsninger.
14	RESULTAT = LØSN 1 + LØSN 2	

2.2. Totalssystemet

Når man skal veje noget på en vægt som den der ses på fig. 2.10, kan man gøre det ved at lægge det, der skal vejes, på den ene vægtskål, og lodder på den anden, så der bliver ligevægt. Vi vil stille os selv følgende opgave:

Hvis man kun vil veje med 1 grams nøjagtighed, hvilke lodder skal man da vælge for at få det færreste antal lodder?

For at kunne veje noget på 1 g må vi naturligvis have et lod på 1 g. En genstand der vejer 2 g kan vejes på to måder: Enten med to lodder på hver 1 g, eller med ét lod på 2 g. Med to lodder på 1 g kan vi kun veje noget, der vejer 1 eller 2 g, men med ét lod á 1 g og ét á 2 g kan vi også veje en genstand på 3 g ved at lægge begge lodder på vægtskålen; vi vælger

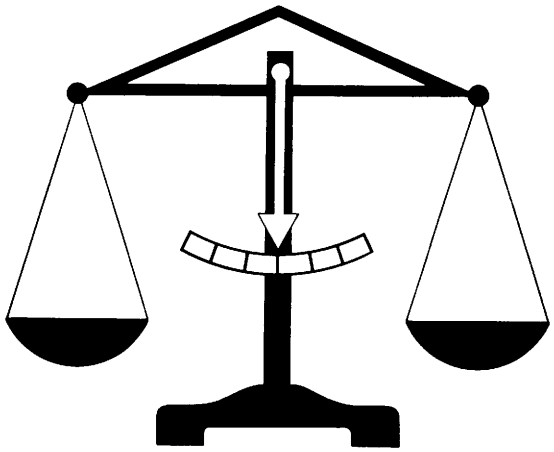


Fig. 2.10. En skålvægt

derfor et lod på 2 g. Hvordan skal vi nu vælge næste lod for at kunne veje noget på 4 g? Vi har de fire muligheder 1 g, 2 g, 3 g og 4 g. Alle disse muligheder kan bruges, men hvis vi vælger et lod på *4 g*, vil vi kunne veje helt op til 7 g (se fig. 2.11). Dette er det bedste vi kan opnå, og vi vælger derfor næste lod som et 4 g-lod. Med de tre lodder, vi foreløbig har, kan vi veje alt fra 1 til 7 g. Mon ikke det næste lod skal være på 8 g? Det kan ikke være større, for så kan vi ikke veje en genstand på 8 g, men hvis det vælges mindre, kan vi ikke dække så mange

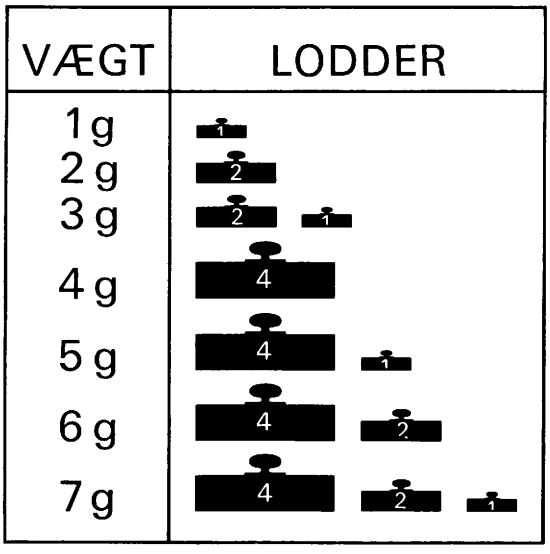


Fig. 2.11. Mulige vejninger med lodder på 1 g, 2 g og 4 g

andre vægte som med et 8 g-lod. Vi vælger derfor et lod på 8 g, og kan nu veje alt mellem 1 og 15 g inkl. Nu kan vi fortsætte på samme måde, og vi finder at det bedste valg af lodder er følgende:

1 g, 2 g, 4 g, 8 g, 16 g, 32 g, 64 g, 128 g, 256 g, 512 g, 1024 g, osv.

Læg mærke til at hvert lod er dobbelt så tungt som det forrige.

Nu skal vi veje noget. Vi lægger f. eks. en fysikbog på den venstre vægtskål og lodder på den højre. Vi begynder med de tungeste lodder, lægger dem på vægtskålen og tager dem af igen, hvis de er for tunge. For at holde regnskab med hvilke lodder vi bruger laver vi et skema med en søjle for hvert lod; hvis et lod skal blive liggende i vægtskålen skriver vi 1 i dets søjle, og hvis loddet er for tungt skriver vi 0.

Forsøg nr.	512	256	128	64	32	16	8	4	2	1
1	0									
2	0	1								
3	0	1	0							
.	.	.	.							
.	.	.	.							
.	.	.	.							
10	0	1	0	1	1	1	0	1	1	0

Loddet på 512 g er for tungt så vi sætter 0 i 512-søjlen (forsøg 1). Så prøver vi med 256 g. Dette lod er for let, så det skal blive liggende i vægtskålen, og vi skal sætte 1 i 256-søjlen (forsøg 2). Med 128 g ekstra bliver vægtskålen med lodder for tung, så må vi fjerne dette lod igen og sætte et 0 i 128-søjlen (forsøg 3). Sådan fortsætter vi med lettere og lettere lodder indtil der er ligevægt, hvilket viser sig at ske når vi lægger 2 g-loddet i vægtskålen. Derfor sætter vi til sidst et 1-tal i 2-søjlen og et 0 i 1-søjlen (forsøg 10).

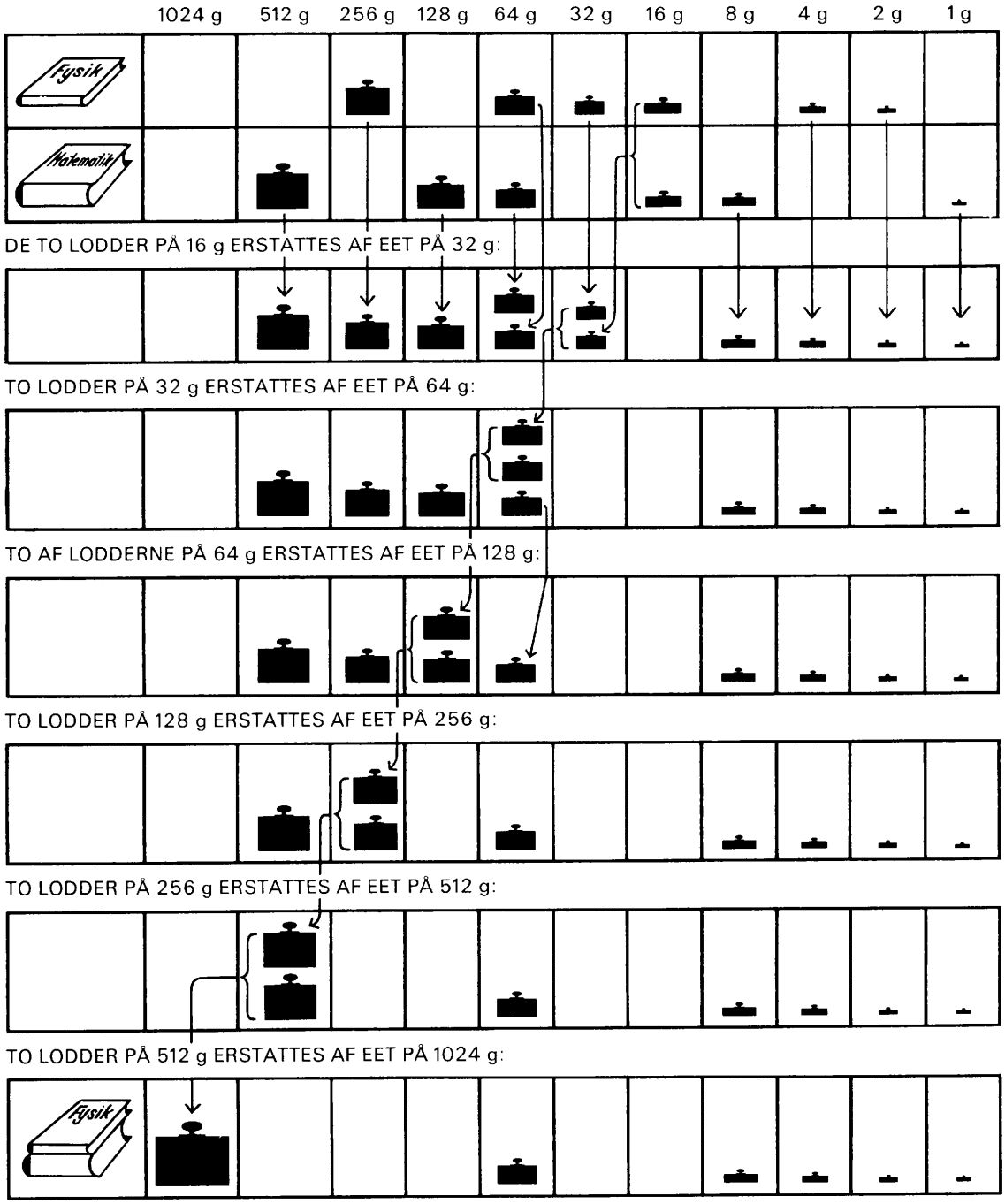


Fig. 2.12. En metode til at bestemme den samlede vægt ud fra vægten af hver del

Fysikbogens vægt kan nu findes ved at vi ser hvor der er 1-taller i skemaet og lægger alle disse vægte sammen. Det bliver 256 + 64 + 32 + 16 + 4 + 2 = 374 g. Denne vægt kan vi også udtrykke som 0 101 110 110, for dette tal fortæller os hvilke lodder vi skal bruge hvis vi vil veje fysikbogen. Når vi kender vejemetoden får vi faktisk det samme at vide om fy-

sikbogens vægt ved at få den opgivet som 374 g, som ved at få den opgivet som lodsamlingen 0 101 110 110.

Lad os nu fjerne fysikbogen og lodderne fra vægten og i stedet veje en matematikbog. Vi finder måske at matematikbogen vejer 1 011 011 001, eller 512 + 128 + 64 + 16 + 8 + 1 = 729 g. Hvis vi vil vide hvad de to bøger vejer *tilsammen* kan vi naturligvis lægge begge bøger på den ene vægtskål og lodder på den anden, men vi kan også forsøge at kombinere resultatet af de to tidligere vejninger. For at gøre det skal vi blot huske at to ens lodder kan erstattes af ét lod som er et »trin« tungere (fordi et lod altid vejer dobbelt så meget som det forrige). Fremgangsmåden ses på fig. 2.12. Vi kan udføre det samme med 1-taller og nuller i stedet for med selve lodderne. Erstatningslodderne kalder vi da for »menten«.

Mente	11 111 1
Fysikbog	0 101 110 110
Matematikbog	1 011 011 001
Samlet vægt	10 001 001 111

Denne metode til kombination af to vægte er – som vi ser – *addition*. De »lodder« vi har at gøre med i form af 1-taller og nuller er i virkeligheden cifre i et talsystem som vi kalder det binære talsystem eller *totalssystemet*. I totalssystemet findes der kun to cifre, 0 og 1. Normalt bruger vi *titalssystemet* som har ti forskellige cifre: 0, 1, 2, 3, 4, 5, 6, 7, 8 og 9. Et tal med tre cifre i titalssystemet, som f. eks. 327, siges at bestå af 3 hundreder, 2 tiere og 7 enere. Man ganger altså cifferets værdi med *10* hver gang man går ét skridt mod venstre. I totalssystemet ganger man med *2* når man går et skridt mod venstre. Tallet 101 i totalssystemet består altså (læst fra venstre mod højre) af 1 firer, 0 toere og 1 ener.

Lad os se om der er mening i vores beregning af bøgernes samlede vægt. Vi fandt at fysikbogen vejede 374 g og at matematikbogen vejede 729 g. Tilsammen må de altså veje 374 + 729 = 1103 g. Beregningen i totalssystemet gav resultatet 10 001 001 111. Hvis vi tager cifrene bagfra (fra højre) og opfatter dem som enere, toere, firere, ottere, ..., tusindogfireogtyvere, får vi:

$$1 + 1 \times 2 + 1 \times 4 + 1 \times 8 + 0 \times 16 + 0 \times 32 + 1 \times 64 + 0 \times 128 + 0 \times 256 + 0 \times 512 + 1 \times 1024 = 1 + 2 + 4 + 8 + 64 + 1024 = 1103.$$

Vi ser at man lige så godt kan regne i totalssystemet som i titalssystemet. Ulempen er at tallene bliver noget længere, men til gengæld skal man kun holde rede på to forskellige cifferværdier.

Man kunne tro at det binære ciffers begrænsning til to muligheder ville indskrænke dets betydning som repræsentant for kendsgerninger eller forestillinger fra det virkelige liv – altså som data – ganske betydeligt. Dette er imidlertid ikke tilfældet. Det viser sig at man netop ofte har brug for at opdele verdens mangfoldighed af muligheder i to grupper: {de sande, de falske}; {de ønskede, de uønskede}; {ja, nej}; {høj, lav}; {mand, kvinde}; {helt, skurk}; {til venstre, til højre}; osv.

Den oplysning – den information – der er i et binært ciffer kaldes *en bit*. Vægten af fysikbogen og matematikbogen tilsammen er altså udtrykt med 11 bit. Når nogen svarer »ja« eller »nej« på et spørgsmål får man en oplysning på 1 bit.

En vigtig egenskab ved totalssystemet er at binære data kan repræsenteres med mekanismer der kun har to tilstande: {nordmagnetisme, sydmagnetisme}; {strøm, ikke-strøm}; {markering, ikke-markering}; {viser til venstre, viser til højre}; osv. – mekanismer der er lette at opfinde

0 0 0 0 0 0	0
0 0 0 0 0 1	1
0 0 0 0 1 0	2
0 0 0 0 1 1	3
0 0 0 1 0 0	4
0 0 0 1 0 1	5
0 0 0 1 1 0	6
0 0 0 1 1 1	7
0 0 1 0 0 0	8
0 0 1 0 0 1	9
0 0 1 0 1 0	10
0 0 1 0 1 1	11
0 0 1 1 0 0	12
0 0 1 1 0 1	13
0 0 1 1 1 0	14
0 0 1 1 1 1	15
0 1 0 0 0 0	16
0 1 0 0 0 1	17
0 1 0 0 1 0	18
0 1 0 0 1 1	19
0 1 0 1 0 0	20
0 1 0 1 0 1	21
0 1 0 1 1 0	22
0 1 0 1 1 1	23
0 1 1 0 0 0	24
0 1 1 0 0 1	25
0 1 1 0 1 0	26
0 1 1 0 1 1	27
0 1 1 1 0 0	28
0 1 1 1 0 1	29
0 1 1 1 1 0	30
0 1 1 1 1 1	31
1 0 0 0 0 0	32
&c.	&c.

bres entiers au-deſſous du double du plus haut degré. Car icy, c'eſt comme ſi on diſoit, par exemple, que 111 ou 7 eſt la ſomme de quatre, de deux & d'un. Et que 1101 ou 13 eſt la ſomme de huit, quatre & un. Cette proprieté ſert aux Eſſayeurs pour peſer toutes ſortes de maſſes avec peu de poids, & pourroit ſervir dans les monnoyes pour donner pluſieurs valeurs avec peu de pieces.

1001	4
10	2
1	1
111	7

1000	8
100	
1	1
1101	13

Cette expreſſion des Nombres étant établie, ſert à faire tres-facilement toutes ſortes d'operations.

Pour *l'Addition* par exemple. ☽

110	6
111	7
1101	13

101	5
1011	11
10000	16

1110	14
10001	17
11111	31

Pour la *Souſtraction.*

1101	13
111	7
110	6

10000	16
1011	11
101	5

11111	31
10001	17
1110	14

Pour la *Multiplication.* ⊙

11	3
11	3
11	
11	
1001	9

101	5
11	3
101	
101	
1111	15

101	5
101	5
101	
1010	
11001	25

Pour la *Diviſion.*

15	1 1 1 1
3	1 1 1 1
	1 1

) 1 0 1 | 5

Et toutes ces operations ſont ſi aiſées, qu'on n'a jamais beſoin de rien eſſayer ni deviner, comme il faut faire dans la diviſion ordinaire. On n'a point beſoin non-plus de rien apprendre par cœur icy, comme il faut faire dans le calcul ordinaire, où il faut ſçavoir, par exemple, que 6 & 7 pris enſemble font 13; & que 5 multiplié par 3 donne 15, ſuivant la Table *d'une fois un eſt un*, qu'on appelle Pythagorique. Mais icy tout cela ſe trouve & ſe prouve de ſource, comme l'on voit dans les exemples précedens ſous les ſignes ☽ & ⊙.

Den tyske matematiker G. W. Leibniz (1646–1716) var den første der indførte de binære tal. Det skete i en afhandling der udkom i året 1703. Her ses en enkelt side fra denne afhandling. Til venstre på siden ses en tabel over binære og decimale tal fra 0 til 32. Midt på siden ses nogle eksempler på regnestykker.

og billige at fremstille med tilfredsstillende driftssikkerhed. Under gennemgangen af kanthulkortet har vi set hvordan kendsgerninger der ikke er binære i deres natur (f. eks. hårfarve) alligevel på effektiv måde kan repræsenteres ved en samling binære cifre (markeringer). Dette er årsagen til at moderne datamater i konstruktion og virkemåde er meget tæt knyttet til den binære datarepræsentation.

OPGAVER

Opgave 2.1.

Giv nogle forslag til registrering af en elevs *vægt* på et kanthulkort, når vægten måles i hele kg. Hvor mange huller skal man mindst bruge ekstra, hvis man ønsker at måle vægten i halve kg?

Opgave 2.2.

Til årets skolekomedie skal bruges 1 stk. heltinde og 1 stk. helt. Disse to personers signalement lyder:

Heltinde: blond, ikke sorte eller grønne øjne.

Helt: lyshåret, blå øjne, højde: mindst 160 cm.

Skriv to programmer til fremfinding af henholdsvis heltinder og helte. Findes der kandidater til rollerne blandt klassens elever?

Opgave 2.3.

Elevregistret ønskes *sorteret* således at elever med mindre legemshøjde står før elever med større legemshøjde. Dette kan gøres på følgende måde: Der stikkes i ethvert af de 9 huller i feltet »højde i cm«. Hullerne stikkes i rækkefølge *bagfra* begyndende med hul nr. 8 og endende med hul nr. 16. Efter hvert stik anbringes kortene på bordet *bag ved* kortene på pinden.

Udfør denne proces og undersøg om den fører til det ønskede resultat. Kan man sortere kortene efter *fødselsmåned* ved brug af et lignende princip (ved stik i hul nr. 61 til 57)? Kan sorteringen gennemføres hvis man ombytter felterne »3« og »4« i det trekantede skema? Prøv at forklare resultatet af disse eksperimenter ved at opskrive hvilke talværdier der svarer til de forskellige måneder, når hul nr. 57 tillægges værdien 1 og hul nr. 58, 59, 60 og 61 tillægges værdierne henholdsvis 7, 4, 2 og 1 (ligesom enerne eller tierne i højden).

Opgave 2.4.

Omregn det binære tal 10 110 til et tal i titalssystemet.

Opgave 2.5.

Omskriv tallet 37 til et binært tal (lad som om en genstand på 37 g skal vejes).

Opgave 2.6.

Fra titalssystemet ved vi at når man trækker fra må man »låne« 10 hvis man skal trække et stort ciffer fra et lille ciffer. I totalssystemet låner man 2 i stedet for 10. Prøv at regne det binære regnestykke:

$$\begin{array}{r} 10\,101 \\ \div\ 1\,011 \\ \hline \\ \hline \end{array}$$

Kontrollér resultatet ved omregning af alle tre tal til titalssystemet.

3. Databehandling og lagring af data

Ved arbejdet med kanthulkort har vi stiftet bekendtskab med to af datalogiens grundlæggende forestillinger, nemlig *datarepræsentationer* og *databehandling*. Når vi taler om datarepræsentationer retter vi opmærksomheden mod den mulighed der er for at vælge forskellige materialer og brugsmåder ved repræsentationen af givne kendsgerninger eller forestillinger. Vi har set at selv når materialet, f. eks. kanthulkortet, er givet, kan den samme forestilling, f. eks. datoen, repræsenteres på mange forskellige måder.

Databehandling er det vi udfører for at få nyttige oplysninger uddraget af de data vi har for hånden. Ved kanthulkort består databehandlingen i at stikke bestemte hulpositioner og at flytte bunker af kort rundt mellem hinanden. Helt almindeligt består databehandling af en række simple handlinger eller *dataprocesser*, som udføres efter hinanden i en ganske bestemt rækkefølge. Som beskrevet i det foregående afsnit kræver selve det at stikke en hulposition først at vi tager kortbunken og strikkepinden, dernæst at vi fører pinden ind i det rigtige hul, dernæst at vi drejer pinden så kortene adskilles, dernæst at vi løfter strikkepinden så de markerede kort frigøres, dernæst at strikkepinden trækkes ud igen, og endelig at de to bunker lægges sammen med andre bunker på en bestemt måde. Selv om dette er det almindelige forløb af en stikkeproces kan der dog optræde variationer afhængigt af særlige omstændigheder. Hvis vi for eksempel opdager at alle kort er markerede i den position der skal stikkes, kan vi helt overspringe de fleste trin i den almindelige stikkeproces.

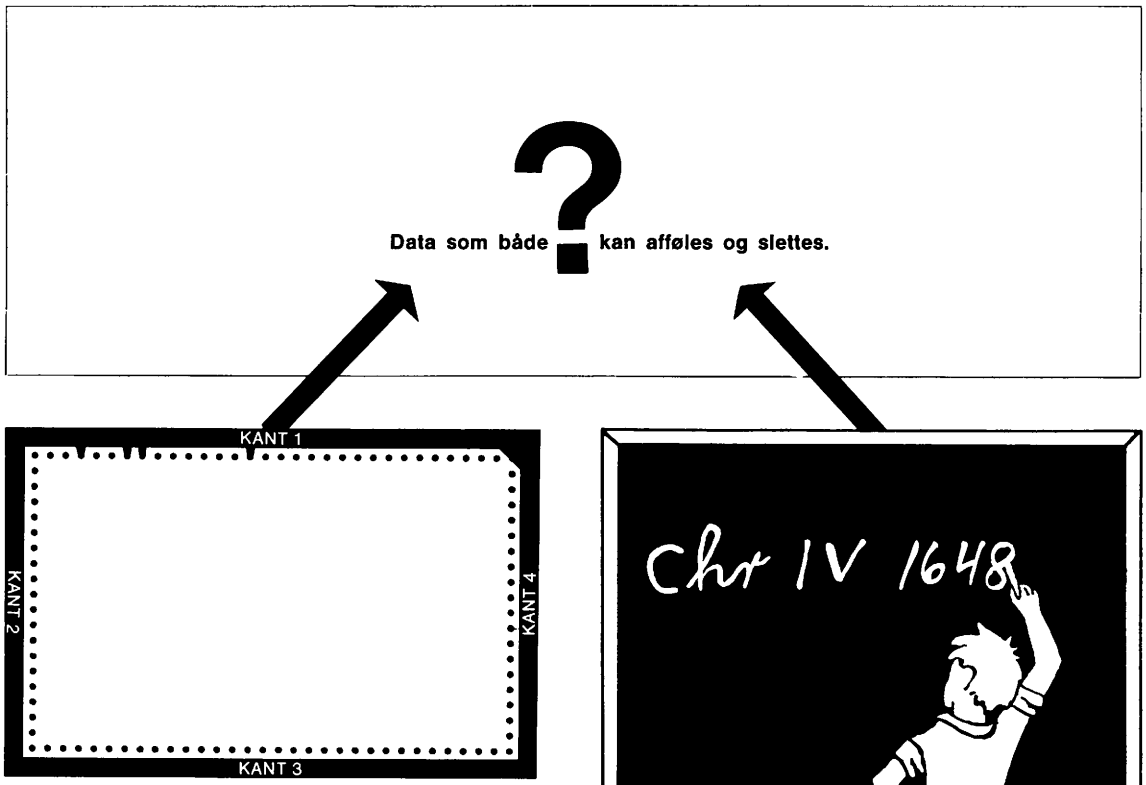
Arbejdet med kanthulkort består nu i at vi udfører en række stikkeprocesser efter et bestemt program. Programmet er en forskrift for netop hvilke stikkeprocesser der skal udføres og i hvilken rækkefølge, og er således selve den måde vi beskriver databehandling på.

Ved arbejdet med kanthulkort er vi stadig i høj grad afhængig af menneskers medvirken. Egentlig er det kun det at aflæse om det enkelte kort er markeret eller ej der er blevet automatiseret. Alt hvad der ellers indgår i en behandling, deriblandt registreringen af data i kortene, udvælgelsen af det hul der skal stikkes, og flytningen af kortbunker, kræver håndarbejde.

For at opnå en højere grad af automatisering af databehandlingen end ved brug af kanthulkort må vi gøre os klart at uanset hvor vidt automatiseringen drives må den allerførste registrering på dataform af fakta og forestillinger der kun kendes af mennesker, f. eks. registreringen af en fødselsdag, nødvendigvis kræve menneskers medvirken. Ligeledes må det færdige resultat bruges af mennesker, ellers tjener behandlingen intet formål. Det der kan automatiseres er mellemstationerne og overgangene mellem dem.

Ved et stik i en bunke kanthulkort sker der en automatisk afføling af de data der er registreret. Derimod egner resultatet af den simple stikkeproces, de to bunker af kort, sig ikke til en fortsat mekanisk behandling. Således er det ikke uden videre muligt uden menneskers medvirken at få markeret netop de kort som er udsøgt gennem en serie af stikkeprocesser. Med andre ord: kanthulkort tillader en mekanisk afføling, men ikke en mekanisk registrering af data. Dette er den afgørende begrænsning ved kanthulkortet.

For at komme videre med automatiseringen



Data som kan afføles mekanisk

Data som kan slettes

af databehandling må vi åbenbart gå på jagt efter datarepræsentationer der både kan afføles og registreres rent mekanisk. Specielt må vi søge efter materialer og brugsmåder der tillader at resultatet af en databehandling fremkommer på samme form som de data der behandles. På den måde vil det nemlig være muligt at automatisere en hel kæde af dataprocesser, derigennem at resultatet fra hvert skridt indgår i den fortsatte behandling helt på linie med de data der blev givet til at begynde med.

Et skridt på vejen mod datarepræsentationer der har denne egenskab er materialer der tillader at data slettes, som f. eks. ved skrivning med kridt på en tavle. Fordelen ved denne art repræsentation er at den samme lagerplads kan bruges om og om igen og at det ikke er nødvendigt at kræve en udvidelse af datakapaciteten hver gang der skal registreres et nyt resultat.

De to repræsentationer, kanthulkort og skoletavle, kan tages som eksempler på to repræsentationer der hver har visse af de egenskaber vi søger, men mangler andre. Kanthulkortet tillader at de registrerede data afføles mekanisk, men resultatet af en proces har ikke samme form som de data der afføles. Skoletavlen tillader at de registrerede data slettes, men affølingen er ikke mekanisk, idet der kræves mennesker til at læse det skrevne.

Det skulle nu være klart at vi til en højere grad af automatisering af dataprocesser behøver mere indviklede datarepræsentationer, der

Blandt datamatudviklingens pionerer er den tyske ingeniør **Konrad Zuse** (f. 1910) den der er startet på det mest beskedne grundlag. Fra 1936 til 1938 byggede han egenhændigt, i et værksted i sine forældres hjem, en automatisk regnemaskine, Z1. Denne var helt igennem mekanisk, baseredes på totalsystemet, og styredes af et program på en hulstrimmel. En ombygget version af Z1, betegnet Z2, hvori visse af de mekaniske dele var erstattet af elektromekaniske relæer, var vidt fremskreden da bygningen blev afbrudt ved krigsudbruddet i 1939. Zuses næste konstruktion, Z3, blev færdiggjort i 1941 og var den første fuldt funktionsdygtige, automatiske, programmérbare regnemaskine i verden. Z3 kunne udføre 15 regneoperationer i sekundet, havde et lager med plads til 64 tal, styredes fra en hulstrimmel, og arbejdede med relæer.

både tillader sletning og mekanisk afføling, og som ydermere gør det muligt at resultatet af en proces registreres direkte som data med samme repræsentation.

Om end vi nu er nået til et punkt hvor de fænomener vi møder i dagliglivet ikke slår til som illustration, så er det dog muligt med simple midler at demonstrere adskillige, forskellige, datarepræsentationer der har disse egenskaber. I den 4. udsendelse skal vi i enkeltheder gennemgå to forskellige datarepræsentationer, den ene rent mekanisk, den anden elektromekanisk.

Som sagt er det ikke muligt at illustrere principperne i automatisk databehandling uden at apparaterne bliver noget indviklede. Det er imidlertid ganske afgørende at man prøver at forstå principperne i apparaterne, og ikke drukner i de mekaniske og elektriske enkeltheder. Vi skal derfor med det samme, som forberedelse til demonstrationen i den 4. udsendelse, gennemgå de principielle træk ved demonstrationerne.

Den første bestanddel i et apparat til automatisk databehandling er et *datalager* (kortere, hvor det ikke kan misforstås: et *lager).* Herved forstår vi noget der kan 1) opbevare dataværdier i ubestemt tid, 2) hvis dataværdier kan afføles mekanisk eller elektrisk så tit det ønskes, uden at indholdet derved forsvinder, og som 3) kan få sine dataværdier ændret gennem mekaniske eller elektriske impulser. Et lager består af et antal *lagerelementer.* Hvert lagerelement kan rumme et *dataelement.* Det simpleste tilfælde er det hvor dataelementerne hver er *en bit,* svarende til at de kun har to mulige værdier. I mere indviklede tilfælde kan lagerelementet rumme et dataelement med flere end to mulige værdier, for eksempel de 28 bogstaver i alfabetet. De enkelte lagerelementer må kunne afføles og få ændret deres dataværdi uden at påvirke hinanden.

Den anden bestanddel er en *impulsgiver,* som er drivkraften for databehandlingen. Den må kunne frembringe en serie af impulser, enten mekaniske eller elektriske, som følger efter hinanden i en bestemt rytme.

Den tredje bestanddel er passende *koblinger* til at overføre impulser mellem datalageret og impulsgiveren. Disse impulser må i første række stamme fra impulsgiveren, men det må være muligt at en impuls gøres afhængig af de data som på et vist tidspunkt befinder sig i datalageret. Tillige må impulserne være af en sådan art at de er i stand til at indsætte nye datavær-

dier i datalageret. Som et vigtigt praktisk krav bør koblingerne være til at ændre, således at databehandlingsapparatet kan omstilles fra én opgave til en anden.

Den fjerde bestanddel af ethvert databehandlingsapparat er et *organ til kontakt* med os mennesker. Vi skal både kunne indsætte dataværdier i datalageret, og kunne aflæse resultaterne af behandlingen.

I de to udformninger der vil blive brugt til illustration er de fire bestanddele:

Mekanisk illustrationsapparat (DRC1)

Datalager: et antal låger der hver kan stå i to forskellige stillinger.

Impulsgiver: brikker der ved deres tyngde bevæger sig afvekslende mellem datalåger, som påvirker deres videre vej, og vippelåger der drejes ved deres passage.

Koblinger: snore, der overfører vippelågernes bevægelse til datalågerne.

Kontaktorgan: datalågerne kan direkte indstilles med håndkraft og aflæses.

Elektromekanisk databehandlingsapparat.

Datalager: et antal relæer, hver med to stillinger. Afhængig af relæets stilling er visse kontakter brudt eller sluttet, hvilket tillader at stillingen afføles elektrisk. Dataværdier indsættes ved at der sendes en elektrisk impuls gennem en af to spoler.

Impulsgiver: strømkilde og kontakter der styres af roterende skiver.

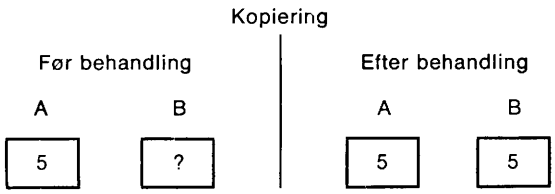
Koblinger: ledninger.

Kontaktorgan: relæernes stilling kan sættes med håndkraft og direkte aflæses.

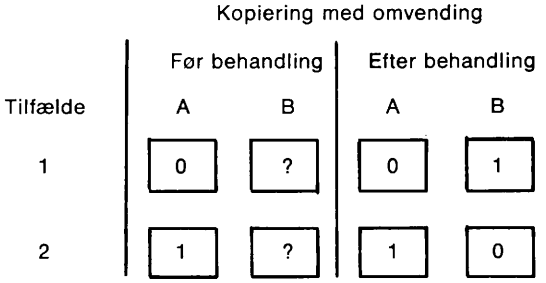
Som det fremgår af den vældige udvikling i brugen af datamater siden 1945, er arbejdsmulighederne for apparater der opbygges omkring disse principper ganske uoverskuelige. For at give et indtryk af den tankegang der gør sig gældende ved dette arbejde skal vi dog ganske kort nævne et par eksempler på simple arbejdsmåder, som indeholder de træk der er karakteristiske for automatisk databehandling.

Den simplest tænkelige databehandling er *kopiering* af en enkelt dataværdi. Kopiering kræver at datalageret har plads til mindst to forskellige dataelementer, A og B. Når kopieringsprocessen sættes i gang kan dataværdierne i A og B være hvad som helst. Når den er afsluttet skal værdien i A være uændret, mens B skal have samme værdi som A. Kopiering er naturligvis ikke særlig nyttig i sig selv. Men ofte har man brug for at omforme dataværdier, uden at miste dem man starter med. Da begynder man med at lave en kopi, som man derefter trygt kan omforme.

Lad os som eksempel på kopiering tænke os at dataelementerne hver kan rumme et af de ti forskellige cifre: 0, 1, 2, . . ., 9. Lad os videre med tegnet ? angive en værdi som kan være hvad som helst uden at påvirke resultatet af behandlingen. Da kan vi opskrive et enkelt eksempel på forløbet af en kopiering således:



Den næstsimpleste databehandling er *kopiering med modifikation*. Hvis dataelementet rummer en enkelt bit kan der kun være tale om en omvending af bitværdien. Igen kan dataværdierne i de to elementer, A og B, før processen være hvilke som helst af de mulige værdier. Efter processen skal A være uændret, mens B skal indeholde den modificerede værdi. I tilfældet hvor A og B hver kun rummer en enkelt bit er der to mulige udfald af behandlingen:



Lidt mere indviklet er den databehandling der ud fra dataværdier givet i to lagerelementer A og B udleder en tredje dataværdi og anbringer den i et lagerelement C. Som et vigtigt eksempel kan nævnes den proces der afgør hvorvidt de to værdier i A og B er ens eller forskellige, også kaldet ækvivalering af A og B. Denne proces skal aflevere værdien 1 (eller ja) hvis A og B er ens, og værdien 0 (eller nej) hvis de er forskellige. Et par illustrationer:

Ækvivalering

	Før behandling			Efter behandling		
Tilfælde	A	B	C	A	B	C
1	3	7	?	3	7	0
2	4	4	?	4	4	1

Endnu mere indviklet bliver det når en vis del af behandlingen afhænger af et resultat der er nået ved en tidligere del af behandlingen. Som eksempel kan nævnes *betinget kopiering:* der er givet fire lagerelementer, A, B, C og D. Såfremt A har samme værdi som B skal C kopieres til D, ellers skal D lades uændret. Til at løse denne opgave kan vi bruge to behandlingstrin, hvoraf det andet kun skal udføres såfremt det første har et givet resultat, nemlig:

Trin 1: Afgør hvorvidt A er lig med B og overfør resultatet til et hjælpedataelement, E.

Trin 2: Såfremt E har værdien 1 (ja), kopier da C's værdi til D.

Illustrationer:

Betinget kopiering

	Før behandling					Efter trin 1					Efter trin 2				
Tilfælde	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
1	2	3	4	5	?	2	3	4	5	0	2	3	4	5	0
2	2	2	4	5	?	2	2	4	5	1	2	2	4	4	1

Opgave 3.1.

Til at ombytte dataværdierne i to dataelementer A og B kræves der tre trin, først at værdien af f. eks. A kopieres til et hjælpedataelement, C, dernæst at værdien af B kopieres til A, og endelig at værdien af C kopieres til B. Vis hvad der skal stå i de tomme rubrikker ved denne databehandling:

	Før behandling			Efter trin 1			Efter trin 2			Efter trin 3		
Tilfælde	A	B	C	A	B	C	A	B	C	A	B	C
1	4	7	?	4	7	4	7	7	4	7	4	4
2	2	5	?									

4. Praktiske forsøg med datalagring

4.1. Indledning

I forrige afsnit omtaltes den meget væsentlige mangel ved kanthulkort og tavle som datalager, nemlig at det ikke er nemt at sætte og afføle dem automatisk. I dette afsnit vil vi gennemgå et meget simpelt mekanisk og et elektromekanisk apparat der kan lagre nogle få dataværdier og udføre simple processer på disse. Disse apparater er sådan udformede at de principielt lige så godt kunne bygges større og dermed udføre mere indviklede beregninger.

4.2. Den mekaniske minimat DRC1

Minimaten DRC1 er en maskine der kan udføre databehandling i miniformat: Når den først er koblet eller »programmeret«, består den menneskelige indsats kun i at putte en brik ind foroven hver gang maskinen skal udføre et procestrin; denne menneskelige indsats kunne imidlertid let erstattes af en passende motoriseret anordning, og dermed ville minimaten blive fuldautomatisk.

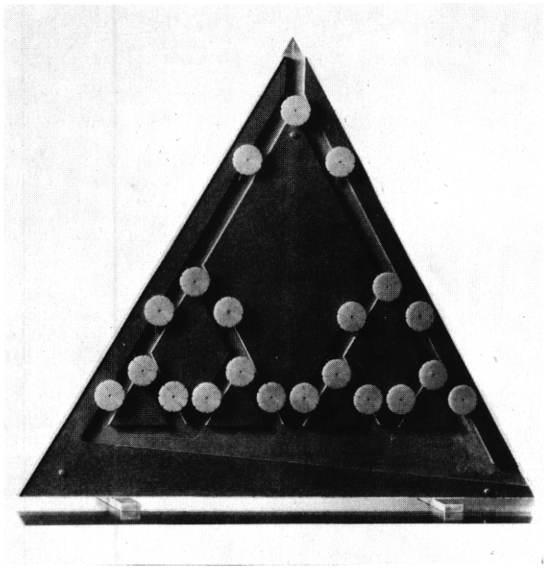


Fig. 4.1. Minimaten DRC1

DRC1 består af et netværk af kanaler hvorigennem en *impulsbrik* kan løbe. I hvert forgreningspunkt af netværket sidder en *datalåge* (betegnet med L1, L2, . . ., L7) og i hver kanal sidder der en *vippelåge* (betegnet med V1, V2, . . ., V14). Skiverne på lågerne kan forbindes med *koblinger* i form af snore.

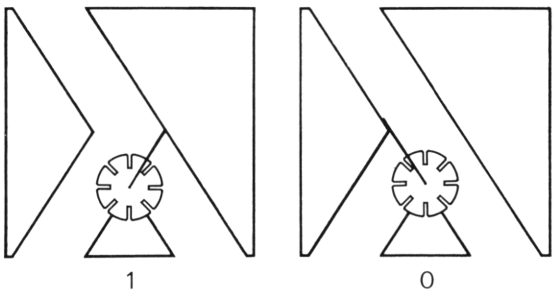


Fig. 4.2. Datalåger set forfra

Datalageret i DRC1 er de 7 datalåger som hver kan stå i netop 2 stillinger. Den ene stilling vil vi betegne som 1, den anden som 0 i overensstemmelse med viserne på bagsiden af DRC1. Hver datalåge kan således lagre en værdi selv om værdien kun kan være 0 eller 1. Som omtalt i udsendelse 2 siger man at hver låge rummer 1 bit. I stedet for at kalde værdierne 0 og 1 kunne vi lige så godt benævne dem NEJ og JA eller FALSK og SAND; det væsentlige er at der er netop 2 mulige stillinger. Fortolkningen af disse afhænger af den brug vi vil gøre deraf, og i de følgende eksempler vil vi dels benytte fortolkningen 0 og 1, dels fortolkningen FALSK og SAND.

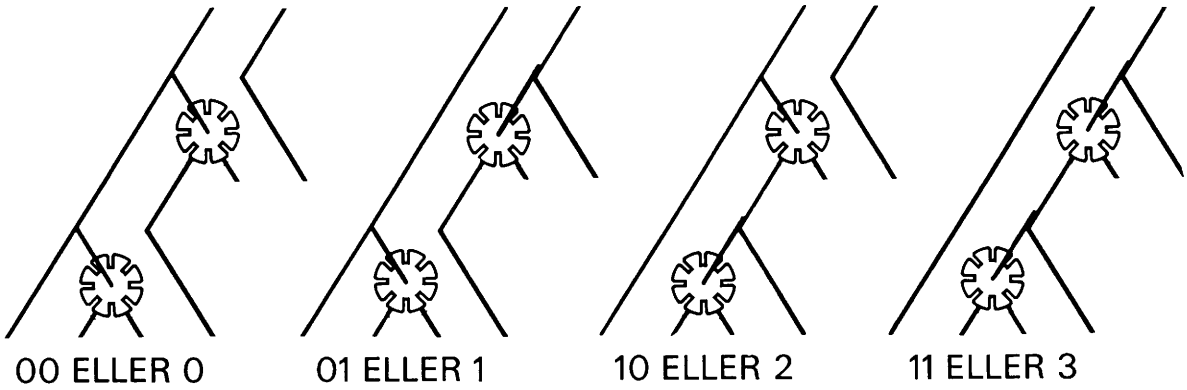


Fig. 4.3. To datalåger tilsammen kan lagre værdierne 0, 1, 2 eller 3

Ved at benytte flere datalåger til hver værdi der skal lagres, kan man udvide værdiområdet på samme måde som når man med kanthulkort benytter flere positioner til én værdi: To datalåger har tilsammen 4 mulige stillinger som f. eks. kan fortolkes som tallene 0, 1, 2 og 3. Tre datalåger har tilsammen 8 mulige stillinger som kan svare til tallene 0, 1, . . ., 7.

En datalåges stilling afføles når impulsbrikken passerer den. I hver kanal under datalågen er der anbragt en vippelåge, som vippes når brikken passerer forbi, og som derfor ved hjælp af koblinger automatisk kan overføre impulser til andre datalåger og eventuelt ændre disses stillinger.

Alt i alt betyder dette, at hver gang impulsbrikken sendes gennem DRC1 udløses der visse funktioner afhængig af datalågernes stilling og af de koblinger vi har sat op, dvs. at DRC1 udfører en dataproces og ender med at være i en ny tilstand, hvor datalågernes nye stilling bestemmer hvad den næste impulsbrik vil udløse. Vi vil nedenfor gennemgå nogle eksempler på hvordan minimaten kan kobles – eller programmeres – til at udføre bestemte dataprocesser.

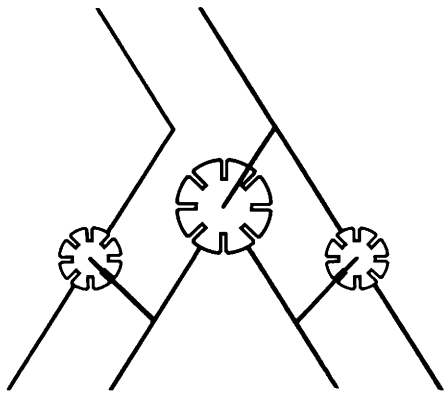


Fig. 4.4. 1 datalåge og 2 vippelåger uden nogen koblinger

Omvending

Vi vil først lave en *omvender*, dvs. et lagerelement der skifter stilling hver gang der sendes en impuls igennem: Står det på 1 skifter det til 0 og omvendt. I DRC1 kan en omvender laves ved at man kobler to nabovippelåger V1 og V2 til datalågen L1 som vist på figur 4.5. På det første billede ses impulsbrikken idet selve affølingen af datalågens gamle stilling (her en 1-stilling) sker; på det andet billede er affølingen forbi, og i dette øjeblik repræsenterer selve brikkens placering den affølte dataværdi; på det tredie billede er brikken endelig ved at skubbe til vippelågen V2 som igen takket være koblingen trækker L1 over i den nye stilling (0). Så snart brikken har passeret vippelågen er det uinteressant hvad der videre sker med brikken; den skal blot løbe ud af apparatet ad en eller anden vej.

Næste gang impulsbrikken sendes ned, løber den forbi V1 som trækker L1 tilbage i 1-stilling, og sådan vil L1 skifte stilling for hver impuls. Dette kan udtrykkes i en meget simpel tabel:

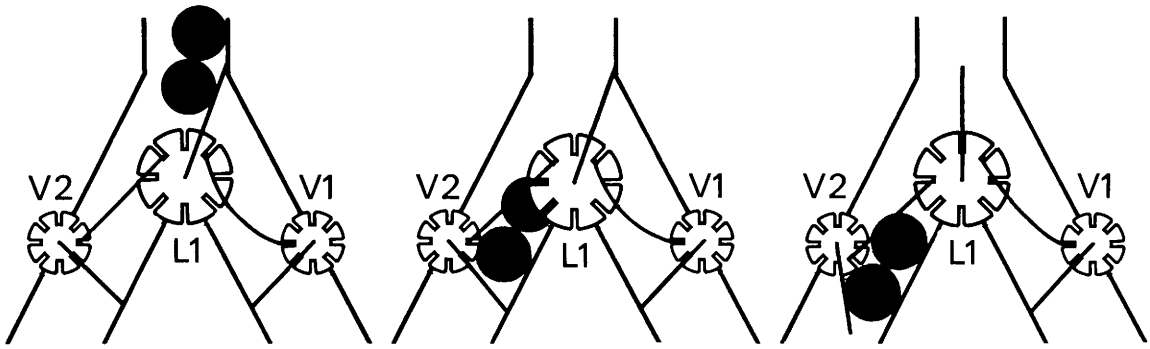


Fig. 4.5. 3 stadier under en omvending fra 1 til 0

Gl. stilling L1	Ny stilling L1
0	1
1	0

Tabel 4.1. Omvending

Kopiering

En anden vigtig funktion er det at kunne kopiere en værdi, der er lagret et sted, over i et andet lagerelement. På figur 4.6 er DRC1 koblet sådan, at når impulsbrikken sendes igennem kopieres værdien af L2 over i L3 eller med andre ord: Når brikken er løbet igennem, har L3 altid samme stilling som L2 uanset hvordan de stod før. L1 skal her altid stå på 0 fordi det altid er L2 der skal afføles.

For at sikre sig at minimaten udfører den ønskede funktion må man prøvekøre den: vi må sende impulsbrikken gennem DRC1 en gang for hver af de forskellige mulige stillinger som L2 og L3 kan have, og så må vi kontrollere at virkningen er den ønskede, altså at lågernes nye stillinger er som ønsket. Da L2 og L3 hver har 2 stillinger er der 4 kombinationer at gennemprøve, og nedenstående tabel viser det ønskede udfald: L2 skal altid være uforandret og L3 skal stå som L2:

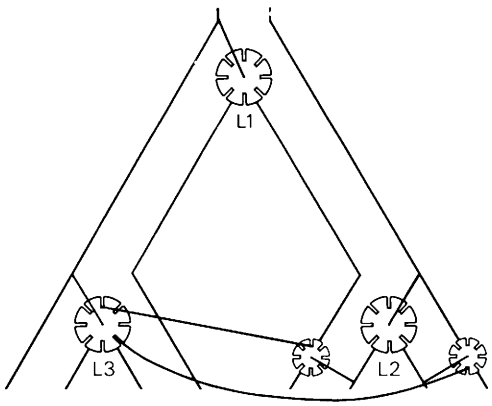


Fig. 4.6. Kopiering fra L2 til L3
(Vippelåger der ikke bruges er ikke vist)

Gl. stilling		Ny stilling	
L2	L3	L2	L3
0	0	0	0
0	1	0	0
1	0	1	1
1	1	1	1

Tabel 4.2. Kopiering

Tælling

Det tredie eksempel skal vise hvordan DRC1 kan kobles som en *tæller*, dvs. sådan at hver gang impulsbrikken sendes igennem, tæller DRC1 et tal 1 op. Hvis den kunne lagre to sædvanlige decimale (titals-)cifre og indstillede vi disse på 00, skulle de for hver impuls ændres til 01, 02, 03, . . ., 09, 10, 11 osv. op til 99.

Vi må imidlertid bruge datalågerne som kun har 2 stillinger hver, og idet vi vælger at bruge L7, L3 og L1 som tre cifre, viser flg. tabel den rækkefølge hvori vi vil tælle disse tre cifre op (samt de tilsvarende decimale talværdier):

Gl. stilling				Ny stilling			
L7	L3	L1	talværdi	L7	L3	L1	talværdi
0	0	0	0	0	0	1	1
0	0	1	1	0	1	0	2
0	1	0	2	0	1	1	3
0	1	1	3	1	0	0	4
1	0	0	4	1	0	1	5
1	0	1	5	1	1	0	6
1	1	0	6	1	1	1	7
1	1	1	7		?		

Tabel 4.3. Tælling

Vi skal imidlertid også fastlægge hvad der skal ske i det sidste tilfælde hvor lågerne står på 111 svarende til det største tal der kan lagres med tre datalåger. Vi vælger her at lave tælleren ligesom kilometertælleren i en bil, dvs. sådan at den fra stillingen 111 vender tilbage til udgangsstillingen 000 (en såkaldt cyklisk tæller).

For at finde ud af hvordan dette skal kobles, bemærker man først at L1 simpelt hen skifter stilling for hver impuls; altså kan den kobles som omvenderen beskrevet ovenfor. Af tabellen kan man finde følgende sammenhæng mellem L1 og L3: Når L1 er 0-stillet skal L3 (og i øvrigt også L7) ikke ændres i næste trin, og når L1 er 1-stillet skal L3 simpelt hen skifte stilling. Altså kan L3 også kobles som en omvender, der, blot fordi den er anbragt under L1's 1-kanal, automatisk spiller korrekt sammen med L1. På lignende vis kan man slutte at det samme gælder L7, og den færdige tæller kan derfor kobles som vist på figur 4.7.

Hvis man ønsker også at registrere hvornår tælleren begynder forfra – dvs. registrere *menten* når man tæller »op« fra 111 til 000 – kan dette ske ved at man f. eks. 1-stiller datalågen L5 i dette tilfælde, og kun da. Det kræver blot en ekstra kobling fra V14 til L5 som vist på figur 4.8.

Lad os følge hvad brikken foretager sig hvis stillingen er 011 som på figur 4.8: Brikken passerer L1 og V2 og 0-stiller derved L1; derefter passerer den L3 og V6 og 0-stiller derved også L3; endelig passerer den L7 og V13 og 1-stiller derved L7, hvorfor slutstillingen bliver 100 som ønsket.

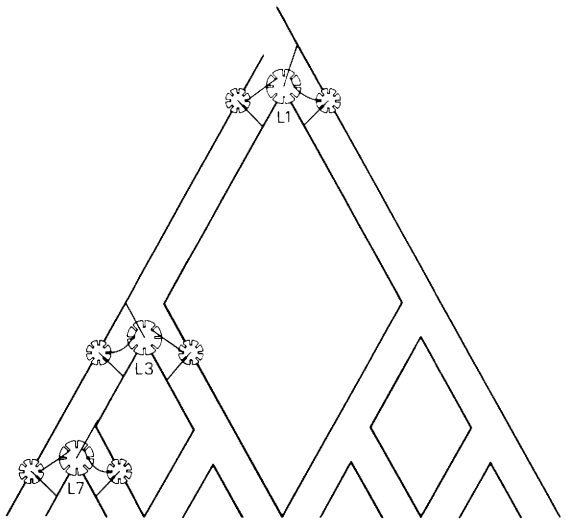


Fig. 4.7. En 3-cifret tæller (kun de låger der er væsentlige er vist. Stillingen af L2, L4, L5 og L6 er helt ligegyldig)

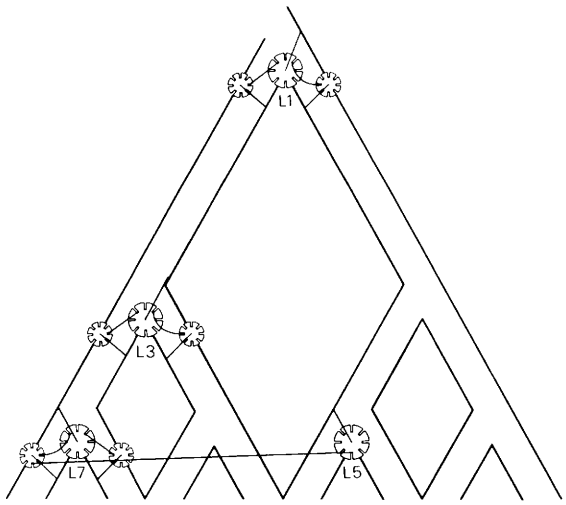


Fig. 4.8. 3-cifret tæller med menteregistrering i L5

På fig. 4.9 ses nogle eksempler på *koblingsskemaer* for DRC1, som det er bekvemt at bruge når man planlægger at bruge DRC1. På skemaet er der plads til (1) at skrive den funktion DRC1 skal udføre, (2) en tabel over lågernes stillinger før og efter en impuls, (3) de startstillinger datalågerne eventuelt skal sættes i, og (4) en skitse af DRC1 hvor man indtegner de ønskede koblinger.

De blanke koblingsskemaer kan bruges til at konstruere nye måder at koble DRC1 på.

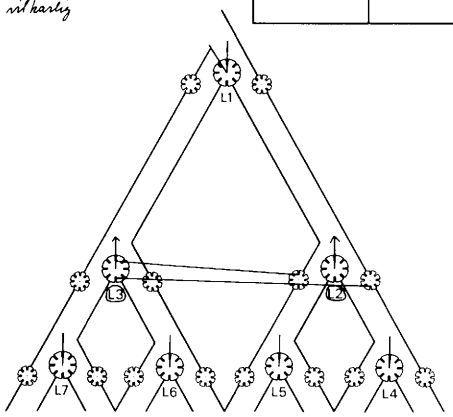
MINIMATEN DRC 1 | KOBLINGSSKEMA cg juli 68 | NAVN: CG DATO: 29-7-68

FUNKTION:
Kopiering: L3 = L2

FUNKTIONSTABEL:

GL. STILLING		NY STILLING	
L2	L3	L2	L3
0	0	0	0
0	1	0	0
1	0	1	1
1	1	1	1

STARTSTILLINGER:
L1: 0
L2: } vilkårlig
L3: }
L4:
L5:
L6:
L7:



MINIMATEN DRC 1 | KOBLINGSSKEMA cg juli 68 | NAVN: CG DATO: 29-7-68

FUNKTION:
1 bits addition:
L3 = ciffer (L1+L3);
L1 = mente (L1+L3);

FUNKTIONSTABEL:

GL. STILLING		NY STILLING	
L1	L3	L1	L3
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

STARTSTILLINGER:
L1: } vilkårlig
~~L2~~:
L3: }
~~L4~~:
~~L5~~:
~~L6~~:
~~L7~~:

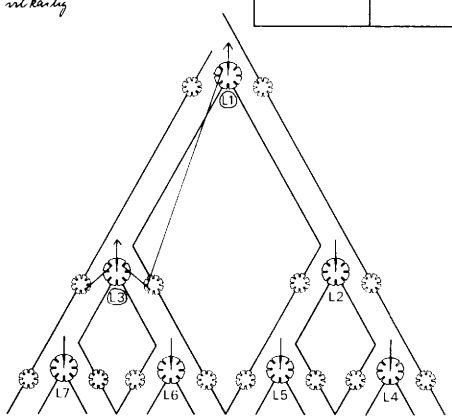


Fig. 4.9. Koblingsskemaer

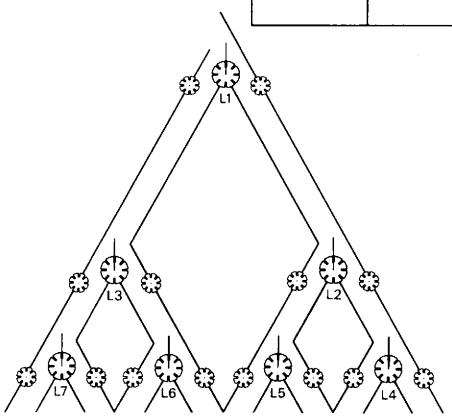
MINIMATEN DRC 1 | KOBLINGSSKEMA | NAVN: DATO:

FUNKTION:

FUNKTIONSTABEL:

GL. STILLING	NY STILLING
L	L

STARTSTILLINGER:
L1:
L2:
L3:
L4:
L5:
L6:
L7:



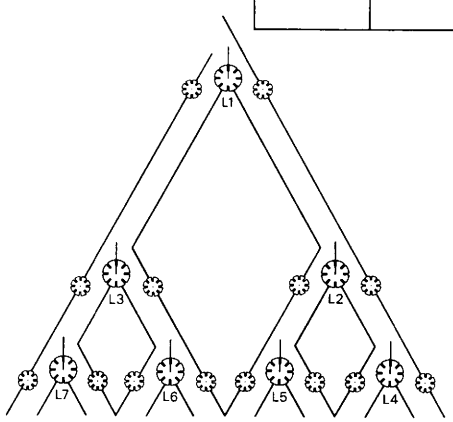
MINIMATEN DRC 1 | KOBLINGSSKEMA | NAVN: DATO:

FUNKTION:

FUNKTIONSTABEL:

GL. STILLING	NY STILLING
L	L

STARTSTILLINGER:
L1:
L2:
L3:
L4:
L5:
L6:
L7:



MINIMATEN DRC 1	KOBLINGSSKEMA	NAVN: DATO:

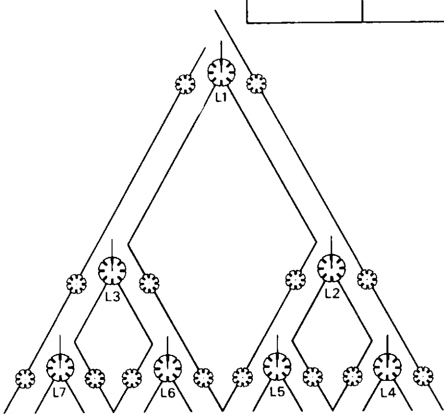
FUNKTION

FUNKTIONSTABEL

GL. STILLING L	NY STILLING L

STARTSTILLINGER

L1
L2
L3
L4
L5
L6
L7:



MINIMATEN DRC 1	KOBLINGSSKEMA	NAVN: DATO:

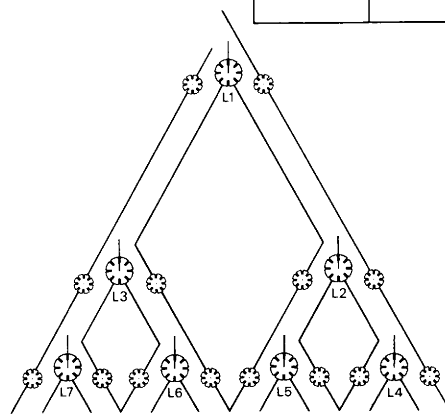
FUNKTION

FUNKTIONSTABEL:

GL. STILLING L	NY STILLING L

STARTSTILLINGER

L1
L2
L3:
L4:
L5:
L6:
L7:



MINIMATEN DRC 1	KOBLINGSSKEMA	NAVN DATO

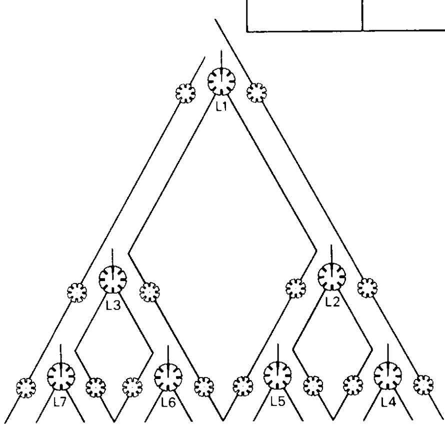
FUNKTION

FUNKTIONSTABEL

GL. STILLING L	NY STILLING L

STARTSTILLINGER

L1
L2
L3
L4
L5
L6
L7



MINIMATEN DRC 1	KOBLINGSSKEMA	NAVN DATO

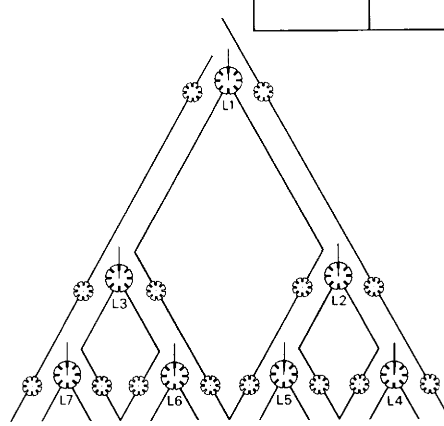
FUNKTION

FUNKTIONSTABEL

GL. STILLING L	NY STILLING L

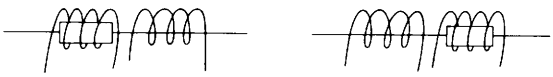
STARTSTILLINGER

L1
L2
L3
L4
L5
L6
L7



Et apparat til databehandling kan opbygges omkring elektromekaniske dele på følgende måde.

Datalager. Dette består af en række elektromekaniske relæer, som hvert har to mulige stillinger og således kan rumme en bit. Hvert relæ har et anker som kan forskydes i sin længderetning under påvirkning fra magnetfelterne i to spoler som omgiver det. Ankeret styrer igennem en forlængelse to elektriske kontakter. Bortset fra de korte tider hvor ankeret er på vej fra den ene yderstilling til den anden vil det kun stå i den ene af de to yderstillinger, som vi kort vil betegne som 0 og 1:



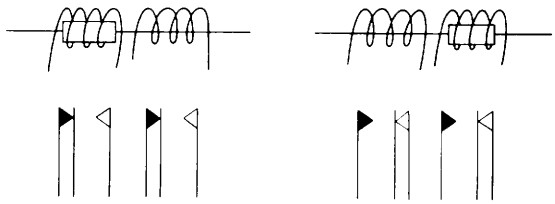
Stilling 0 Stilling 1

For at bringe relæet i stilling 0 må vi sende en elektrisk impuls gennem den venstre spole. Stilling 1 nås med en impuls gennem den højre spole.

Relæer kan fås med flere forskellige kontaktarter. Vi vil bygge på relæer der hver er udstyret med to skiftekontakter, dvs. kontakter der er opbygget som antydet i følgende signatur:

Skiftekontakt

Denne signatur betegner en kontakt med tre forbindelser, hvor den midterste leder er sluttet til den sorte kontaktspids når relæet står i stilling 0, men til den hvide når relæet står i stilling 1. Et fuldstændigt relæ der er udstyret med to skiftekontakter kan tegnes i sine to stillinger således:

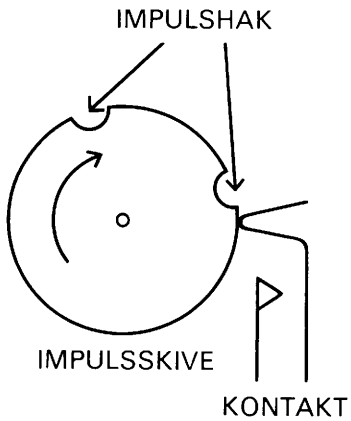


Stilling 0 Stilling 1

Når vi tegner kredsløb med relæer vil vi ikke tegne begge stillinger, men kun stilling 0. Signaturerne vil da tillade os at slutte os til hvorledes forbindelserne er i stilling 1.

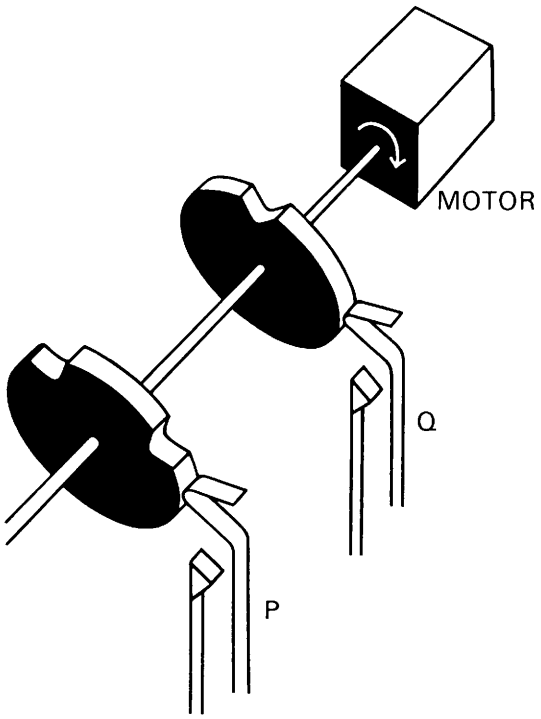
At et relæ med denne opbygning kan anvendes som element i et datalager fremgår af (1) at ankeret vil *forblive* i den ene eller den anden af sine to stillinger, og relæet således kan *opbevare* sin dataværdi, så længe man ønsker det, (2) at relæets stilling, og derigennem den dataværdi elementet rummer, kan *afføles* ved at der sendes strømimpulser *gennem kontakterne,* og (3) at relæets stilling, og derigennem dataværdien, kan *ændres* ved at der sendes en elektrisk impuls gennem *den ene eller den anden af de to spoler.*

Impulsgiveren i vores apparat opbygges af en strømkilde, en række elektriske kontakter, og en roterende aksel der bærer en impulsskive for hver kontakt. En impulsskive og den kontakt den styrer ser således ud:



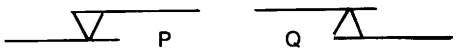
Hver gang et impulshak passerer forbi kontakten vil denne kortvarigt sluttes.

En impulsgiver med to kontakter P og Q er opbygget således:

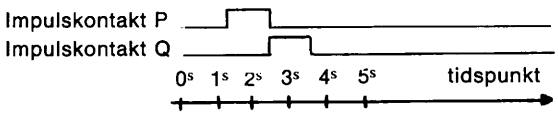


Ved at anbringe impulshakkene og impulsskiverne passende i forhold til hinanden er det muligt at få P og Q til at sluttes og afbrydes i enhver tænkelig rækkefølge.

Til at vise impulsgiverens virkemåde i en bestemt opstilling vil vi ikke tegne selve skiverne, men blot kontakterne og tidspunkterne når de er sluttet og afbrudt. Kontakterne tegnes som sluttekontakter, f. eks.

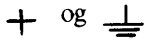


Til at angive hver kontakts slutten og afbryden tegnes en linie der følger tiden fra venstre mod højre og som springer opad når kontakten sluttes og nedad når den afbrydes. Således angiver for eksempel følgende:



at P først, og Q lidt efter er sluttet et kort øjeblik. Har vi brug for at tale om bestemte tidspunkter under forløbet kan de angives således:

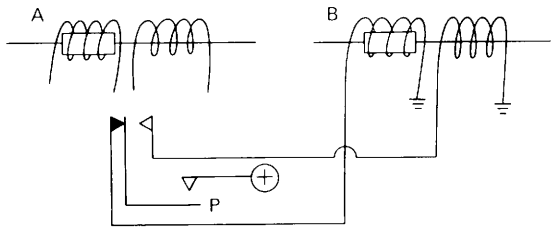
Som strømkilde kræves et batteri eller en passende forbindelse til lysnettet. På kredsløbstegningerne angives strømkildens to poler med



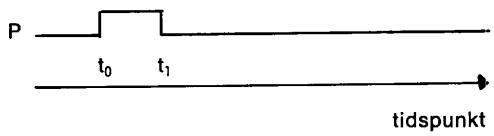
Koblingerne besørges af elektriske ledninger. For at ændre apparatets funktion er det nødvendigt at flytte impulsledningerne og at ændre impulsgiverens skiver.

Som *kontaktorganer* tjener forlængelser af relæernes ankre, som både kan bevæges og aflæses direkte.

Kopiering af en enkelt bitværdi fra A til B realiseres med disse elektromekaniske midler gennem følgende kredsløb, som kun udnytter den ene af A's kontakter:



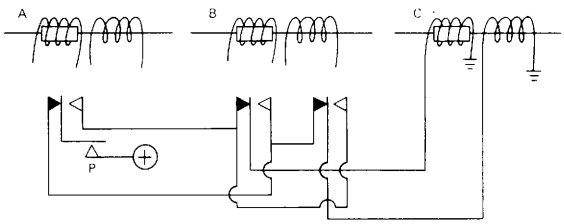
Her er P en kontakt i impulsgiveren der styres således:



Kopieringsprocessen sættes i gang til tiden t_0 og er afsluttet til tiden t_1.

For at overbevise os om at dette kredsløb virker efter hensigten må vi efterprøve de fire mulige udgangsstillinger for relæerne A og B. Vi vil da konstatere at den ene af B's spoler altid udsættes for en strøm, men i to af de fire tilfælde vil B's anker allerede stå i den ønskede stilling og derfor ikke bevæge sig.

Et kredsløb til at sætte C til 0 hvis værdierne i A og B er forskellige og til 1 hvis de er ens (dvs. til at overføre ækvivalensen af A og B til C), kan opbygges således:

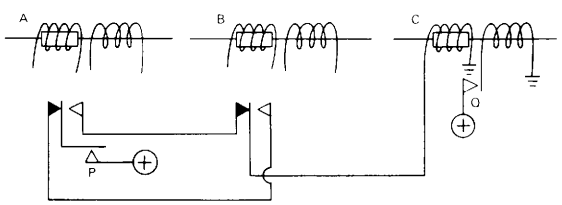


P er igen en kontakt i impulsgiveren der skal sluttes en kort periode når processen skal udføres.

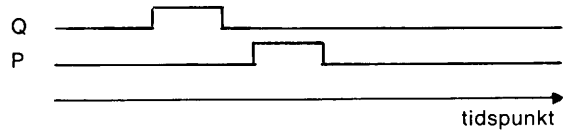
For at kontrollere kredsløbet er det denne gang nødvendigt at overveje 8 mulige udgangsstillinger for de tre relæer A, B og C. Disse stillinger, og relæernes stilling efter processen vises i følgende tabel:

Startstilling nr.	Før proces			Efter proces		
	A	B	C	A	B	C
1	0	0	0	0	0	1
2	0	0	1	0	0	1
3	0	1	0	0	1	0
4	0	1	1	0	1	0
5	1	0	0	1	0	0
6	1	0	1	1	0	0
7	1	1	0	1	1	1
8	1	1	1	1	1	1

Det gælder som en vigtig almindelig regel i datalogien, at selv når bestemte hjælpemidler til databehandlingen er givet – som i dette tilfælde vore relæer, osv. – vil der findes flere forskellige måder at løse samme opgave på. Dette kan vi illustrere ved nedenstående kredsløb, der viser en anden måde at overføre ækvivalensen af A og B til C:



Der kræves nu to kontakter i impulsgiveren, P og Q, der skal styres således:



I denne løsning sørger Q for at C til at begynde med sættes til 1. Derefter sættes C tilbage til 0 såfremt A og B indeholder forskellige dataværdier. Denne løsning er derfor langsommere end den første og kræver ydermere kontakten Q. Til gengæld spares den ene skiftekontakt i B.

OPGAVER

Opgave 4.1.

Hvor mange datalåger er nødvendige for at kunne lagre tallene 0–99? Tallene 0–999?

Opgave 4.2.

Lad DRC1 være koblet som på figur 4.6. Lav den til tabel 4.2 svarende resultattabel hvis V3 og V4 også kobles til L2 som en omvender. Aflæs af tabellen hvad der sker hvis L2 og L3 står ens før impulsbrikken kommer, samt hvad der sker hvis de står modsat.

Opgave 4.3.

Betragt tælleren i figur 4.8. Lav en kobling som nulstiller menten i datalåge L5 samtidig med tællingen fra stillingen 011 til 100.

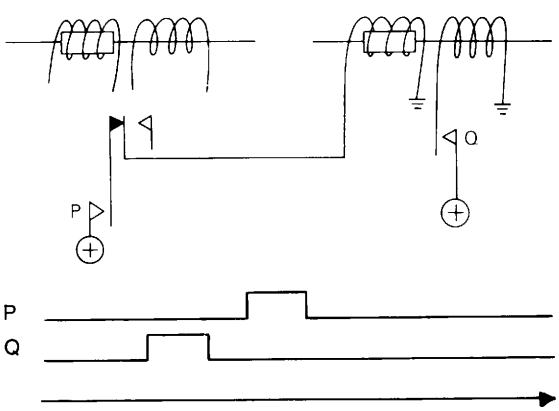
Opgave 4.4.

Lad datalågernes stilling repræsentere SAND og FALSK i stedet for 1 og 0. Konstruér de koblinger der sætter L3 til SAND hvis L1 eller L2 eller begge er SAND, og ellers FALSK (L3 får værdien af udsagnet »L1 eller L2« – dette kaldes også *disjunktionen* af L1 og L2).

Hvilket udsagn repræsenterer L3, hvis vi lader lågernes stillinger bytte mening, så 1 svarer til FALSK og 0 til SAND?

Opgave 4.5.

Der er givet følgende kredsløb:

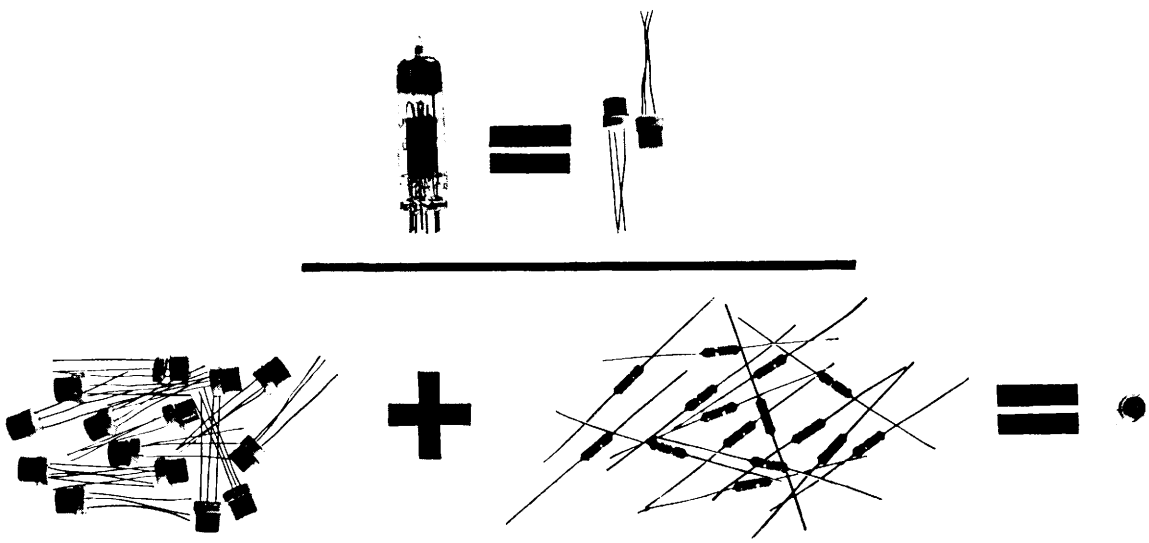


Find ud af hvilken databehandling det udfører.

Opgave 4.6.

Tegn kredsløbet for en elektromekanisk opstilling der overfører værdien af A $\wedge$ B (læses: konjunktionen af A og B) til C, efter følgende forskrift:

A	B	A$\wedge$B
0	0	0
0	1	0
1	0	0
1	1	1



Den tekniske udvikling af datamater viser en kraftigere og kraftigere formindskelse af de enkelte komponenter. Oprindelig benyttede man elektronrør, der senere afløstes af transistorer (en udvikling der også kendes fra f. eks. radiomodtagere), men i den nyeste tid er man begyndt at anvende de såkaldt integrerede halvlederkredse, hvor en enkelt komponent erstatter et helt transistorkredsløb med ca. 30 transistorer og modstande. Billedet giver et indtryk af størrelsesforholdene mellem komponenter med samme ydeevne.

5. Programmering af datamater

5.1. Lagrede programmer.

De store elektroniske datamater fungerer principielt fuldstændig som minimaten DRC1 og som de relækredsløb der blev omtalt i 4. udsendelse, men der er tre væsentlige årsager til den store forskel i ydeevne.

For det første er der ingen mekanisk bevægelige dele i selve datamaten, idet de elektromekaniske relæer er erstattet af rent elektroniske kredsløb der kan udføre de samme funktioner som relæerne. Herved opnår man dels at maskinen bliver langt mere driftssikker fordi der er langt mindre »slid« på elektronik end på mekanik, dels at maskinen arbejder med en uhyre meget større hastighed: Mens et meget fint mekanisk relæ har en reaktionstid på ca. $^1/_{100}$ sekund, kan et elektronisk relæ som det bygges i dag reagere henved en million gange så hurtigt, altså på ca. $^1/_{100000000}$ sekund.

For det andet indeholder en elektronisk datamat langt flere dele end vi arbejdede med i 4. udsendelse. Mens DRC1 har 7 datalåger rummer en stor datamat adskillige millioner tilsvarende elektroniske dele. Det ville imidlertid være umuligt at få en maskine af den størrelse til at fungere hvis den var bygget af mekaniske dele eller elektromekaniske relæer, fordi disse ikke er tilstrækkeligt driftssikre. Derfor kan så store maskiner kun fungere hvis de er bygget udelukkende med elektroniske dele.

For det tredie sker programmeringen af en datamat ikke ved at man påsætter snore eller trækker ledninger fordi det antal ledninger der skulle trækkes er så stort at det ville være umuligt at holde styr på dem.

I DRC1 programmerede vi et enkelt programtrin ad gangen: F. eks. svarer snorekoblingerne for den 3-cifrede cykliske tæller til at vi sætter DRC1 i stand til at udføre en bestemt ordre, nemlig denne tælling. Ved at sende impulsbrikken ned flere gange kan vi selvfølgelig få udført denne ordre flere gange, men det er den *samme* ordre hver gang. I en virkelig databehandlingsopgave skal der imidlertid som det fremgik af 2. og 3. udsendelse udføres mange forskellige ordrer efter hinanden, ofte mange tusinde programtrin, og dette ville kræve et mylder af ledninger. Derfor styres en datamat af lagrede programmer: For hver opgave datamaten løser bruges dens lager dels til de *data* der skal behandles i denne opgave, dels til det *program* der fastlægger den ønskede behandling af disse data; ved et program forstås som tidligere omtalt alle de ønskede ordrer i den ønskede rækkefølge. For hver ny opgave som datamaten skal løse starter man derfor med at anbringe de dertil hørende data og det ønskede program i lageret; så først er datamaten klar.

Et program består af en række ordrer, en for hvert programtrin, anbragt i lageret i den rækkefølge de skal udføres. Hver ordre anbringes som en bestemt indstilling af et bestemt antal lagerelementer. Datamatens *styreenhed,* der bl. a. også fungerer som en slags drivmotor, begynder med at udsende impulser der afføler hvad det første sæt lagerelementer indeholder, dvs. hvad den første ordre i programmet er. Denne ordreafføling får styreenheden til at udsende impulser der udfører det beordrede programtrin på de data der findes andetsteds i lageret. Når den første ordre således er udført, udsender styreenheden automatisk impulser der afføler det næste sæt lagerelementer, dvs. den næste ordre i programmet; dette får styreenheden til at udføre det dertil svarende programtrin osv.

Da et program kan bestå af mange ordrer, og da det skal kunne indlæses og lagres i en data-

mat, er det nødvendigt med et bekvemt og meget præcist og systematisk sprog til nedskrivning af programmer; et sådant sprog, som altså er en samling regler for hvordan man skriver ordrer og sætter dem sammen til programmer, kaldes et *programmeringssprog*. I næste paragraf viser vi et eksempel på et sådant programmeringssprog.

Rundt om i verden – og også i Danmark – bruges mange forskellige programmeringssprog til de forskellige datamater, og disse sprog afviger en hel del fra hinanden.

Egentlig kræver hver datamatmodel sit eget specielle programmeringssprog, og da der er flere hundrede forskellige modeller, er der lige så mange af disse *maskinorienterede* programmeringssprog. Det gør det vanskeligt at udveksle programmer og erfaringer, og derfor opstod i midten af 1950'erne den idé at skrive programmerne i et fælles (såkaldt *problemorienteret)* sprog, og så lade hver datamat oversætte et program til sit private sprog før den kan gå i gang med at udføre programmet. Denne teknik er nu højt udviklet og meget anvendt, og der er lavet adskillige sådanne almindelige programmeringssprog hvoraf i hvert fald fire har vundet meget stor udbredelse:

Fortran (*For*mula *Tran*slator, dvs. formel-oversætter) var det første der slog an; det blev udviklet af det store amerikanske datamat-firma IBM til brug på egne datamater og bruges stadig mange steder i verden, især i USA. *Algol* (*Algo*rithmic *L*anguage, dvs. algoritmisk sprog, altså et sprog til at beskrive algoritmer eller beregningsmetoder) er udviklet af en international komité og bruges meget, navnlig i Europa, inklusive Østlandene. I Algol er de grundlæggende begreber stærkt generaliseret i forhold til Fortran og sproget er gjort helt uafhængigt af specielle datamatmodellers egenskaber. Mens Fortran og Algol især er velegnede til at beskrive tekniske beregningsopgaver, er *Cobol* (*Co*mmon *B*usiness *O*riented *L*anguage, dvs. fælles forretningsorienteret sprog) navnlig tænkt til administrative og kontormæssige databehandlingsopgaver; det blev udviklet og understøttet af de amerikanske offentlige myndigheder. Endelig er *PL/1* (*P*rogramming *L*anguage One, dvs. programmeringssprog 1) udviklet af IBM inden for de seneste år og er tænkt at skulle afløse de øvrige. Det er derfor et meget omfattende sprog med mange flere forskellige udtryksmuligheder end de andre sprog. Det er imidlertid stadig et åbent spørgsmål hvor stor udbredelse PL/1 vil få.

I de følgende eksempler på programmering vil vi bruge Algol fordi det er det mest konsekvente og veldefinerede sprog og fordi det er det sprog der bruges hyppigst i tidsskrifter og bøger vedrørende programmering.

5.2. Programmeringssproget Algol

I dette afsnit vil vi – uden at gå for meget i enkeltheder og uden at tilstræbe en fuldstændig definition af begreberne – vise nogle af grundtrækkene i Algol og nogle eksempler på programstumper.

Navne og lagring

En central proces i ethvert program er at lagre en foreløbig beregnet dataværdi, for senere at kunne hente den frem igen og bruge den. I Algol kan en lagringsordre f. eks. se således ud:

(5.1) $\quad A := 2;$

(ordren læses: »A får værdien 2«)
som betyder at værdien 2 anbringes et sted i lageret under navnet A sådan at når denne værdi skal indgå i en beregning senere i det samme program, bruger vi blot navnet A i den ønskede sammenhæng. F. eks. vil den kombinerede beregnings- og lagringsordre

(5.2) $\quad B := A + 7.5;$

(ordren læses: »B får værdien A plus 7 komma 5«)
derefter bevirke at

1. værdien af $A + 7^1/_2$ beregnes, med resultatet $9^1/_2$,
2. denne værdi anbringes et nyt sted i lageret under navnet B.

På højre side af : = kan man skrive vilkårligt komplicerede beregningsudtryk på næsten samme måde som vi er vant til i matematik. F. eks. kan arealet af en cirkel med radius B–4.1 beregnes med de tre ordrer:

(5.3)
```
Pi := 3.1416;
r := B - 4.1;
Areal := Pi × r↑2;
```

som bevirker at

```
5160 NYV1(I2) = V2(I3)
5200 CONTINUE

     DO 8000  I4 = 2, 8

     DO 5203  L1 = 1, 45
5203 TEKSTLIN(L1)= 0
     LINDEL   = 15
     LINIETIL = 2
     I5       = I1 + I4 -1
     NRCELLE  = I4 -1
```

Stump af Fortran program

```
ks64 := 0;
for i := 1 step 1 until part do
  begin comment udvælg de partier, der skal have tillægsmandater;
    k := 0;
    for j := 1,t2,t3 do if SPO[i,j]>SOL[j] then k := k + 1;
    if sample[i + t14]>0∨k>t2∨sample[i]>gyld stem/50
     then ks64 := ks64 + sample[i] else sample[i] := 0
  end undersøgelse af om parti skal have tillægsmandat;
```

Stump af Algol program

```
        02  PMNT PICTURE ZZZZ9 .
PROCEDURE DIVISION.
START.
    OPEN INPUT AJOUR-KORT, OUTPUT AJOUR-BAAND, TRYK.
    MOVE ZERO TO S.
    PERFORM LAES-KORT.
    MOVE DWN IN STYREKORT TO DWN IN LIN-1.
    MOVE OS-HOVED TO A IN LIN-1, MOVE ZERO TO SIDETAELLER,
    MOVE 99 TO LINIETAELLER, PERFORM LINIETAELLING.
    IF LOEBENR-G EQ '9' THEN
        DISPLAY 'INTET GAMMELT KARTOTEK, ER ALT MON OK';
```

Stump af Cobol program

```
WEATHER:PROCEDURE;
     DECLARE MAXDAY (7),MINDAY(7), AVERAGE (7);
     READ LIST ((MAXDAY (I),MINDAY (I)) I=1 TO 7);
     AVERAGE = (MAXDAY + MINDAY)/2;
     WRITE ((MAXDAY (I), MINDAY (I),AVERAGE (I)) I=1 TO 7)
               (2F(5),F(8,1),SPACE);
END WEATHER;
```

Stump af PL/1 program

Fig. 5.1. Eksempler på programmer

1. værdien 3.1416 lagres under navnet Pi,
2. værdien B – 4.1 = 5.4 beregnes og lagres under navnet r,
3. med ovennævnte værdier for Pi og r beregnes Pi×r² og lagres under navnet Areal. Bemærk at potensopløftning skrives med tegnet ↑ i stedet for »løftede« tal.

Hvis størrelserne Pi og r ikke skal bruges andre steder i samme opgave – samme program – kunne man lige så godt skrive arealberegningen som én ordre

(5.4) Areal := 3.1416 × (B – 4.1)↑2;

men hvis værdierne bruges adskillige gange, er det bekvemt at give dem navne. De navne man vil benytte for de forskellige størrelser i et program kan sammensættes frit af bogstaver og cifre blot det første tegn i hvert navn er et bogstav.

En værdi der er lagret under et eller andet navn kan slettes og erstattes af en ny værdi blot ved at man skriver en ny lagringsordre med det samme navn på venstre side af :=. Således bevirker de 4 ordrer

(5.5)
```
n := 2;
B := n↑n;
n := 3;
C := n↑n;
```

at

1. n får værdien 2
2. B får værdien $2^2 = 4$
3. n får værdien 3
4. C får værdien $3^3 = 27$, altså en anden værdi end B selv om selve beregningsudtrykket n↑n er helt det samme som i den anden ordre.

I ovenstående eksempel kunne den 3. ordre også være skrevet således:

(5.6) n := n + 1;

fordi beregningen altid udføres før lagringen, og virkningen er derfor

1. Med den gamle værdi af n, dvs. 2, beregnes værdien af n + 1, altså 3,
2. den beregnede værdi 3 lagres som n (og n's gamle værdi slettes automatisk).

Repetition og indicering

I enhver realistisk databehandlingsopgave indgår der beregninger der skal gentages mange gange med kun få og små ændringer fra gang til gang. Derfor indeholder Algol en repetitionsordre som bevirker at en eller flere beregningsordrer kan repeteres et vist antal gange med små ændringer.

Lad os illustrere brugen heraf med et eksempel: I stedet for som ovenfor at beregne 2^2 og 3^3 vil vi gerne lave en tabel over f. eks. 1^1, 2^2, 3^3, . . ., 15^{15}. En simpel, men ufiks måde at gøre dette på ville være at skrive et langt beregningsprogram i stil med (5.5):

(5.7)
```
n := 1;
A := n↑n;
n := n + 1;
B := n↑n;
n := n + 1;
C := n↑n;
---
n := n + 1;
O := n↑n;
```

Programmet viser helt tydeligt at den egentlige beregningsproces er den samme alle 15 gange blot med den lille ændring at n øges med 1 for hver gang. Derfor kan man i stedet bruge repetitionsmekanismen

for n := 1 *step* 1 *until* 15 *do* – – –

efterfulgt af den beregnings- og lagringsordre der skal repeteres. Dette betyder at først udføres denne ordre mens n har værdien 1 (takket være *for* n := 1 . . .); så udføres den med n = 2 (takket være . . . *step* 1 . . .), så med n = 3 (også takket være . . . *step* 1 . . .) osv., indtil den sidste gang udføres med n-værdien 15 (takket være . . . *until* 15 . . .).

Den ordre der skal stå efter *do* skal se omtrent således ud:

```
---:= n↑n;
```

men hvad skal navnet på venstre side af := være? Hvis vi skriver A := n↑n bevirker det at først får A den rigtige værdi, men derefter lagres efterhånden *alle* værdierne under *samme* navn og sletter derfor hinanden ud efterhånden; når sætningen er blevet repeteret de 15 gange, er $A = 15^{15}$ og alle de øvrige værdier er gået tabt. Det tilsvarende vil ske lige meget hvilket navn vi vælger at skrive. Det er derfor nødvendigt også at have en anden måde at skrive navne på sådan at et navn efter ønske kan dække over, eller snarere pege på, forskellige medlemmer i et vist sæt af navne. I matematikken bruges ofte den teknik at sætte indeks på navnene sådan at man lader f. eks.

$$T_1, T_2, T_3, \ldots$$

betegne beslægtede størrelser – det kan være punkter i en geometrisk opgave eller ubekendte i en beregningsopgave. På helt tilsvarende måde kan man i Algol bruge lignende *indicerede navne*, blot skal selve indeks skrives i en kantet parentes i stedet for som et lille »fodtal«:

```
T[1], T[2], T[3], ...
```

Hvis man f. eks. i et program bruger de 15 forskellige indicerede navne T[1], ..., T[15], betyder det at der gøres plads til 15 værdier i lageret, og hver enkelt af disse kan indgå i beregninger eller få lagret en ny værdi nøjagtig som om man havde valgt 15 forskellige af de tidligere benyttede (simple eller ikke-indicerede) navne.

I vores tabelproblem kan vi derfor prøve at skrive

```
for n := 1 step 1 until 15 do
  T[ ] := n↑n;
```

Som indeks kan vi ikke bruge 1 for så ville alle værdierne stadig blive lagret oven i hinanden under navnet T[1]. Indeks skal variere i takt med repetitionerne og derfor skal der stå

(5.8)
```
for n := 1 step 1 until 15 do
  T[n] := n↑n;
```

Virkningen er nu:

1. Først får n værdien 1 og beregningsordren udføres, hvorved T[1] får værdien $1^1 = 1$.
2. Derefter øges n med 1 og beregningsordren udføres, hvorved den næste størrelse T[2] får værdien $2^2 = 4$.
3. Dette spil gentages og slutter med at den sidste størrelse T[15] får værdien 15^{15}. Når dette er sket er alle de 15 beregnede værdier lagret og kan senere i programmet bruges efter ønske.

Bemærk at i sætningen T[n] := n↑n bruges størrelsen n i to betydninger: I beregningsudtrykket n↑n er n en talværdi der indgår i beregningen, men i det indicerede navn T[n] er n en viser der udpeger et bestemt navn i samlingen T[1], ..., T[15]. Denne dobbelte brug af heltallene kendes også fra matematikken hvor de både bruges som egentlige talværdier i formler og som indeks til at ordne samlinger af ubekendte og lignende.

Vi har hidtil talt om *navne* – både simple og indicerede – på steder i lageret og om de *værdier* der lagres på disse steder. Det vil imidlertid være bekvemt at have en fællesbetegnelse for et lagerelement, det tilknyttede navn og den just i øjeblikket lagrede værdi. Man bruger *variabel* – simpel eller indiceret – som betegnelse for alt dette, og vi vil bruge udtryk som »Variablen A får værdien ...«, »Den indicerede variabel T[n] har nu værdien ...« og lignende.

Betingede ordrer og hopordrer

I databehandlingsopgaver er det nødvendigt at kunne skrive ordrer der kun skal udføres hvis et tidligere beregnet resultat opfylder en bestemt betingelse. En sådan ordre kaldes en *betinget ordre* og den udtrykker et valg mellem *to* muligheder: Hvis den betingelse, der undersøges, er opfyldt udfører maskinen én ting, ellers en anden. Med andre ord bestemmer svaret på den opstillede betingelse hvad der videre skal ske.

Vi vil først som eksempel på brugen heraf tage den samme tabelopgave som ovenfor, men med den ændring at tabellen ikke længere skal gå fra 1 til 15 men skal stoppe når resultatet n^n bliver større end 1 million. Ved hjælp af logaritmeregning kunne man måske nok finde ud af hvor langt n så må gå, og derefter kunne man bruge program (5.8) blot med den nye slutværdi for n i stedet for 15. Men hvis vi ikke vil have besværet med at finde denne slutværdi på forhånd – og i mange andre tilfælde er det praktisk umuligt – kan vi ikke bruge repetitionsmekanismen

for n := *step until do*

fordi slutværdien skal optræde mellem *until* og *do*. I stedet vil vi skrive et program – eller rettere en programstump – som mere direkte svarer til opgavebeskrivelsen: Beregn en tabel over 1^1, 2^2, 3^3, ... indtil resultatet n^n bliver større end 1 million.

Da vi ikke kan bruge repetitionsmekanismen til at styre hvilke værdier n skal gennemløbe, må vi vende tilbage til at bruge ordrer der minder om program (5.7):

```
n := 1;
T[n] := n↑n;
n := n + 1;
```

For at kunne tale om de enkelte ordrer i dette program vil vi sætte navne (etiketter) på hver sætning, og vi vælger at bruge etiketterne 01, 02 og 03:

```
(5.10)  01:  n := 1;
        02:  T[n] := n↑n;
        03:  n := n + 1;
```

Det ændrer intet ved programmets funktion men gør det muligt at henvise til de enkelte ordrer.

Vi vil nu tilføje en ny ordre efter (5.10), som skal betyde at nogle af ordrerne repeteres indtil n^n er blevet større end 1 million. Vi skal nemlig blot repetere ordrerne 02 og 03 for at få beregnet den ønskede tabel. Betingelsen for at vi skal repetere skrives således:

if n↑n < 1 000 000 *then* – – –

hvor der efter *then* skal stå en eller anden ordre. Meningen med dette er at sålænge n^n er mindre end 1 000 000 udføres ordren efter *then*, og derfor skal dette være en ordre om at vende tilbage og gentage de to ordrer 02 og 03. En ordre om at vende tilbage – en *hopordre* – skrives således:

go to – – –

hvor man efter *go to* skriver etiketten på den første af de ordrer der skal repeteres. I vort tilfælde er det ordren 02, og derfor bliver hele programmet

```
        01:  n := 1;
(5.11)  02:  T[n] := n↑n;
        03:  n := n + 1;
        04:  if n↑n < 1 000 000 then go to 02;
```

Når dette program startes, sker der følgende i de første trin:

Trin	Udfør sætning	n får værdien	øvrige virkninger
1	01	1	
2	02		T[1] får værdien $1^1 = 1$
3	03	2	
4	04		da $2^2 < 1000000$, fortsættes med 02
5	02		T[2] får værdien $2^2 = 4$
6	03	3	
7	04		da $3^3 < 1000000$, fortsættes med 02
8	02		T[3] får værdien $3^3 = 27$

osv. indtil n^n på et eller andet trin er blevet større end (eller lig med) 1 million; når datamaten så kommer til 04, udføres hopordren *go to* 02 ikke, og dermed er programstumpen (5.11) udtømt. Hvis den er en del af et større program, fortsætter datamaten med at udføre de ordrer der følger umiddelbart efter 04.

I det sidste eksempel vil vi kombinere brugen af repetitionsordrer og betingede ordrer, og eksemplet drejer sig om *søgning*. I dagliglivet er

man ustandselig udsat for at skulle udføre en søgning, f. eks. hver gang man slår op i en telefonbog eller i en ordbog, eller når man bladder i en avis for at finde en bestemt rubrik. Søgning forekommer også meget hyppigt som led i en databehandling, og vi vil prøve at skrive et program for følgende problem som man må forestille sig blot er en lille del af en større databehandlingsopgave: Der er givet 500 tal, som f. eks. kan repræsentere vægten af en række personer; undersøg om der blandt disse er én med vægten w, hvor w også er et opgivet tal.

Den simpleste måde at løse opgaven på er at gennemgå de 500 tal fra en ende, undersøge for hvert tal om dets værdi er lig med w, og hvis dette er tilfældet stoppe den videre søgning. Omsat til Algol betyder det at de 500 vægttal på en eller anden måde må være lagret som et sæt indicerede variable, og lad os kalde dem

Vægt[1], Vægt[2], ..., Vægt[500].

Vi vil uden at forklare det nærmere antage at denne lagring allerede er sket, ligesom vi vil antage at den vægtværdi vi skal undersøge forekomsten af er lagret i en variabel w.

Det at gennemgå de 500 variable fra en ende af kan gøres med repetitionsmekanismen

for n := 1 *step* 1 *until* 500 *do* ...

Hvis det kun er en enkelt ordre der skal repeteres, skrives denne blot efter *do* som i det foregående eksempel. Men hvis repetitionen skal omfatte flere ordrer, må disse omgives af *begin* og *end* der virker som en slags parenteser:

```
for n := 1 step 1 until 500 do
  begin
  ...
  ...
  end;
```

betyder at først sættes n = 1 og alle ordrerne mellem *begin* og *end* udføres; så sættes n = 2 og alle ordrerne udføres en gang til; således fortsættes indtil ordrerne udføres sidste gang med n = 500.

Mellem *begin* og *end* skal vi derfor skrive det der skal repeteres 500 gange, altså selve undersøgelsen af om et vægttal er lig med w. Undersøgelsen kan få to udfald: Hvis det vægttal vi undersøger *er* lig med w skal vi stoppe, ellers skal vi fortsætte søgningen; dette kan udtrykkes med en betinget ordre

if ... *then* ...

Undersøgelsen skal skrives mellem *if* og *then* og må her bestå i et spørgsmål af typen

if Vægt [ ] = w *then* ...

I den kantede parentes skal der skrives et nummer (et indeks) som gør at alle 500 vægttal gennemløbes, når spørgsmålet bruges i forbindelse med repetitionsmekanismen. Men dér vil den variable n netop gennemløbe de ønskede indeks-værdier, og vi kan derfor skrive

```
for n := 1 step 1 until 500 do
  begin
  if Vægt [n] = w then ...
  end;
```
(5.12)

For at finde ud af hvad der mere må stå i programmet vil vi forestille os at vi har sat datamaten i gang med at udføre program (5.12) og at den er midt i undersøgelsen, f. eks. sådan at den netop er færdig med nr. 16. Det næste der sker er at n får værdien 17 og datamaten undersøger om Vægt[17] er lig med w: (1) Hvis svaret er nej, overspringes ordren lige efter *do* (endnu står der kun tre prikker) og maskinen fortsætter med næste ordre. Men dette er *end* som betyder at denne omgang af repetitionen er færdig, og derfor gentages spillet med n = 18. (2) Hvis svaret er ja, udføres ordren lige efter *do*, og dette skal altså være en ordre om at afbryde undersøgelsen fordi svaret ja betyder at der er fundet et vægttal med den ønskede værdi. Dette kan gøres med en hopordre

go to JA;

hvor JA er navnet (etiketten) på en ordre længere nede i programmet. Det kan f. eks. være en

ordre om at udskrive nummeret på det fundne vægttal på datamatens kontaktorgan – det er netop værdien af n i det øjeblik undersøgelsen har givet svaret ja – eller det kunne være den første af de ordrer som databehandlingsopgaven skal fortsætte med.

Programmet bliver derfor:

```
        for n := 1 step 1 until 500 do
          begin
(5.13)    if Vægt [n] = w then go to JA;
          end;
        ...
    JA: udskriv (n);
```

Når man skriver et program for en databehandlingsopgave må man imidlertid sikre sig at man har gennemtænkt *alle* muligheder og at programmet fungerer efter hensigten i *alle* tilfælde. Her er der i hvert fald to tilfælde der kræver særlig omtanke, nemlig (1) at der er *flere* vægttal blandt de 500 som har værdien w, og (2) at der *ingen* vægttal findes af den ønskede størrelse. Lad os undersøge hvad program (5.13) vil gøre i disse specielle tilfælde:

(1) Hvis der er flere af vægttallene som er lig w – lad os f. eks. antage at Vægt[13] = Vægt[22] = Vægt[23] = w – vil programmet blot finde det første (her altså nr. 13) og slet ikke »opdage« de øvrige fordi undersøgelsen stoppes så snart det første er fundet. Hvorvidt dette er ønskeligt eller ej må helt og holdent afhænge af hvad resultatet videre skal bruges til, og vi vil ikke gøre mere ved dette tilfælde her.

(2) Hvis derimod ingen af vægttallene er lig w sker der følgende: Repetitionen får lov til at løbe helt til ende uden at svaret på spørgsmålet Vægt [n] = w er ja en eneste gang, og derfor er hopordren *go to* JA ikke blevet udført. Efter at det sidste spørgsmål Vægt[500] = w også har givet svaret nej, fortsætter datamaten derfor med at udføre den ordre (eller de ordrer) der følger efter *end* i repetitionsmekanismen. I program (5.13) er det de tre prikker mellem *end* og JA, og de skal altså erstattes af en eller flere ordre der giver besked om at søgningen er mislykkedes. I (5.13) hvor nummeret på det fundne vægttal udskrives, kan vi f. eks. erstatte prikkerne med en lagringsordre der giver n en let kendelig værdi – som f. eks. –1 eller 1000 – og derpå lade programmet fortsætte som i (5.13) med at udskrive n. Det endelige program bliver på denne måde:

```
        for n := 1 step 1 until 500 do
          begin
(5.14)    if Vægt[n] = w then go to JA;
          end;
        n := -1;
    JA: udskriv (n);
```

Under denne gennemgang af et programmeringssprog har vi slet ikke omtalt hvordan de nødvendige data kommer ind og ud af datamatens lager. *Indlæsning* af de dataværdier programmet skal bruge og *udskrivning* af resultaterne skal programmeres lige så vel som de egentlige beregninger, og det sker ved hjælp af ordrer der styrer datamatens kontaktorganer.

De mest benyttede kontaktorganer til indlæsning er apparater der kan aflæse hulkort eller hulstrimler, og de dataværdier der skal indlæses må derfor først overføres til hulkort eller hulstrimmel på specielle skrivemaskiner. På figur 5.2 ses et *hulkort;* det er inddelt i en række kolonner der hver rummer et tegn hvad enten dette er et ciffer, et bogstav eller et af de andre tegn. Hvert tegn har sin bestemte kombination af huller som hulles i en kolonne når man trykker på den tilsvarende tangent i tastaturet på hullemaskinen. Samtidig skriver maskinen tegnet i sædvanlig skrift oven over kolonnen, men dette sker kun for at gøre det nemmere at kontrollere at værdierne er hullet korrekt. Datamatens kontaktorgan aflæser kun hulkombinationen, og skriften foroven er derfor overflødig i forbindelse med selve indlæsningen.

En *hulstrimmel,* som ses på figur 5.3, indeholder ingen skrift, men ellers er princippet i datarepræsentationen det samme som for hulkort: Hver kolonne på strimlen rummer et tegn, og hvert tegn har sin bestemte hulkombination som hulles når man trykker på den tilsvarende tast på skrivemaskinen. For kontrollens skyld skriver maskinen samtidigt med hulningen alting på papir ganske som en almindelig skrivemaskine.

De mest benyttede kontaktorganer til udskrivning af resultater er meget hurtige skrivemaskiner som direkte skriver de ønskede resultater i sædvanlig klar skrift – hvis blot programmet indeholder de rigtige udskrive-ordrer på de rigtige steder.

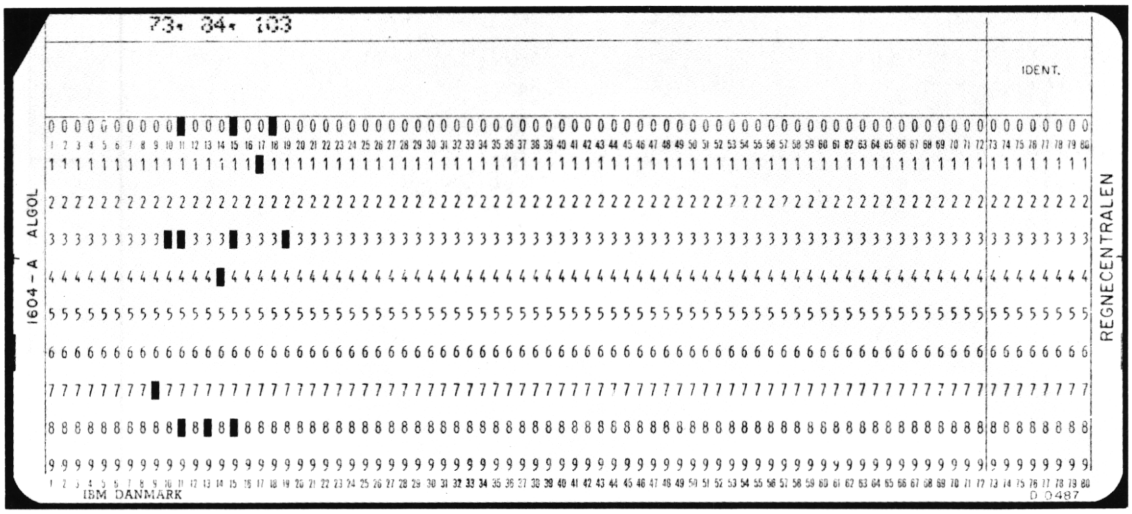


Fig. 5.2. Et hulkort med dataværdierne 73, 84 og 103

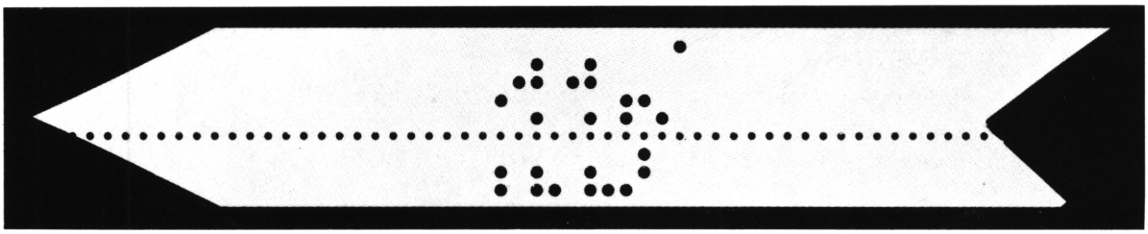


Fig. 5.3. En stump af en hulstrimmel

Hvis man f. eks. vil prøve at bruge program (5.14) i praksis, mangler der nogle ordrer i starten til at sørge for at de 500 vægttal bliver indlæst og anbragt i lageret før ordrerne i (5.14) kan udføres. Sidst i programmet har vi antydet at værdien af n skal udskrives, men i virkeligheden kræves der lidt mere udførlige ordrer for at fortælle mere præcist hvad der skal udskrives og hvordan det skal ske.

Under gennemgangen har vi heller ikke omtalt hvordan selve programmet bliver anbragt i lageret før det overhovedet kan udføres. Dette kræver for det første at programmet udstyres med endnu nogle ordrer til indledning og til afslutning, og dernæst at også programmet skrives på hulkort eller hulstrimmel ganske ligesom de dataværdier der skal indlæses. Som regel skal man altså lave både en programstrimmel og en datastrimmel – eller tilsvarende hulkort – før datamaten kan startes. Men hele arbejdsgangen for at løse en opgave på en datamat vender vi i øvrigt tilbage til i 6. udsendelse.

OPGAVER

Opgave 5.1.

Lav en lille tabel over hvilke værdier de variable N og Sum får under udførelsen af flg. programstump:

```
Sum := 0;
for N := 1 step 1 until 9 do
  Sum := Sum + N;
```

Opgave 5.2.

Forklar hvad følgende program udretter:

```
      s := 0;
      i := 0;
L3:   i := i + 1;
      s := s + i;
      M[i] := s;
      if s < 20 then go to L3;
```

10	42	19	36	59	52	49	2	57	62	13	33
19	31	62	51	59	10	4	15	47	56	37	39
19	31	62	51	59	10	6	15	47	37	57	39
43	38	48	10	55	8	61	54	59	25	17	33
43	47	36	10	55	8	61	54	59	25	17	33
51	11	36	10	28	56	40	4	59	50	62	23
51	11	36	10	28	57	40	6	59	50	62	23
54	9	36	28	18	56	40	4	59	50	62	23
54	9	36	28	18	57	40	6	59	50	62	23
57	19	52	10	14	63	59	49	42	35	4	33
63	59	19	12	57	10	7	49	51	37	40	33

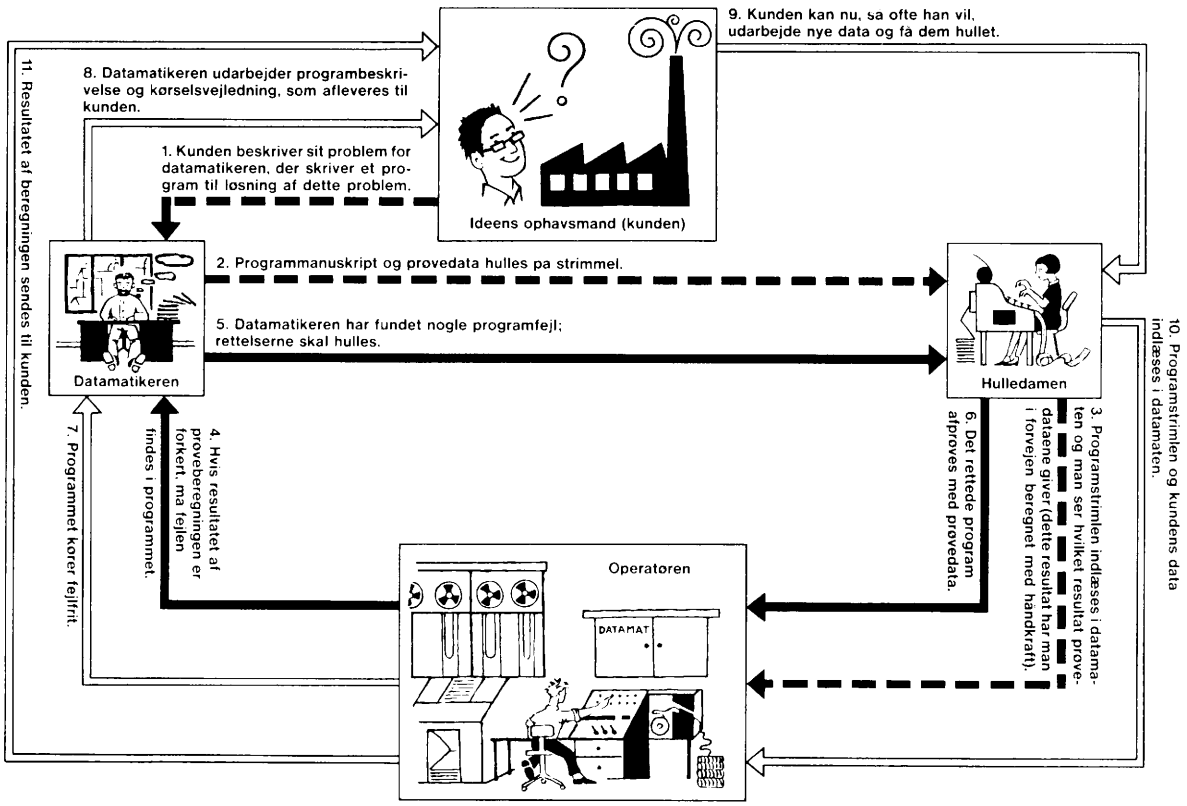
I skolen lærer vi at vi skal skrive navn og klasse uden på vore stilehæfter, altså at der må lægges vægt på at medtage en ramme af data omkring det der er den egentlige sag, nemlig de danske stile. Dette princip bevarer sin gyldighed i alt arbejde med data. Vi viser her et dårligt, afskrækkende eksempel på hvordan det vi får en datamat til at levere os **ikke** skal se ud. Som tallene står, uden forklaring, er de værdiløse, nonsens.

6. Datamatikerens arbejde

6.1. Arbejdsgangen ved løsning af en databehandlingsopgave

Pilene på figuren skal læses i nummerorden. Løsningsprocessen former sig som en rundgående bevægelse mellem datamatikeren, hulledamen og operatøren. Disse tre personer er kun en del af det personale der betjener datamaten. Andre funktioner som f. eks. teknisk vedligeholdelse og modtagelse og forsendelse af data varetages af personer der for overskuelighedens skyld ikke er vist på figuren. Datamatikeren deltager i planlægningen og programmeringen af opgaven; undertiden udfører han denne del af arbejdet alene (som på figuren), men ofte varetages det af flere personer, idet det mere planlæggende arbejde udføres af nogle personer og selve programmeringen af andre. Man kan skelne mellem tre faser i arbejdet:

1. *Problemformulering og programmering,* hvor opgaven defineres og formuleres i et programmeringssprog. Denne fase er på figuren angivet ved hjælp af punkterede pile.
2. *Prøvekørsel* (sorte pile) hvor uundgåelige fejl og misforståelser findes og rettes. Prøverundturen på figuren må ofte gennemløbes adskillige gange før programmet er fejlfrit.
3. *Rutinekørsel* (hvide pile) hvor det fejlfrie program benyttes om og om igen med nye data, f. eks. nye koefficienter til et program der kan løse ligninger.



Kunden, datamatikeren, hulledamen og operatøren er betegnelser for fire forskellige delfunktioner i hele arbejdsgangen. Hvis det drejer sig om en forholdsvis lille databehandlingsopgave vil nogle af disse funktioner ofte i praksis blive udført af én og samme person (måske kan kunden selv programmere, eller datamatikeren huller selv sit program og betjener selv datamaten).

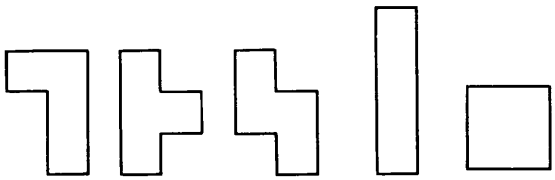
6.2. Løsning af Tetromino-puslespil.

For at give et indtryk af datamatikerens arbejde med at formulere problemer således at de kan løses af en datamat skal vi gennemgå problemformuleringen for løsningen af den puslespil-opgave der er stillet i tillægget bag i hæftet. Opgaven er at lægge tolv pentominobrikker således at de netop udfylder et givet område.

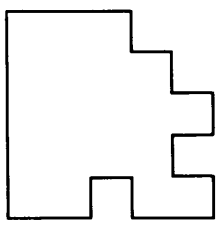
Det må siges med det samme at opgaven er så indviklet at vi kun vil berøre visse mere væsentlige træk ved den automatiske løsning. Vi skal koncentrere os om hvorledes det overhovedet er muligt at automatisere søgningen efter en løsning til puslespillet, idet netop dette antagelig vil være det mest uforståelige for den der kun har søgt at løse opgaven på den sædvanlige, halvt tilfældige måde der benyttes af mennesker.

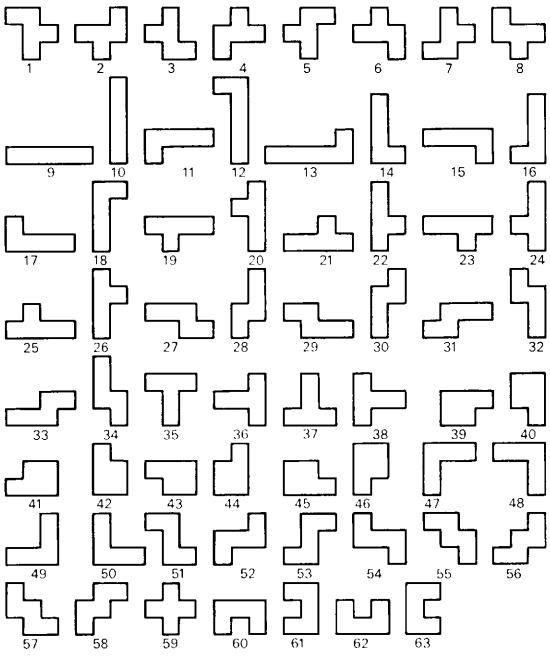
Det afgørende ved en automatisk løsning med datamat er at der må være system i tingene. Den tilfældige måde hvorpå mennesker forsøger sig frem ved lægningen af et puslespil kan ikke overføres til et program. Der gælder imidlertid ikke det omvendte. Den systematik der kræves ved en automatisk løsning kan såre vel bruges også af mennesker. Imidlertid vil formentlig derved charmen ved opgaven gå tabt, fordi det hele bliver for mekanisk. Men for at forklare den automatiske løsning vil vi vise hvordan den systematiske metode kunne benyttes også uden brug af en datamat.

Den systematiske metode vi skal gennemgå kan bruges på en hel klasse af opgaver, ikke blot på den bestemte opgave der er stillet i tillægget. Således kan brikkerne se ud som de vil, blot de er opbygget af lige store kvadrater, og også brættet kan have en vilkårlig form så længe det er sammensat af så mange lige store kvadrater som brikkerne kan dække. Fremgangsmåden er principielt den samme i alle tilfælde, og for at gøre det hele mere overskueligt vil vi i det følgende se på en simplere opgave, hvor brikkerne er de fem tetrominoer (dvs. brikker der hver består af fire kvadrater):



og hvor brættet har følgende form:

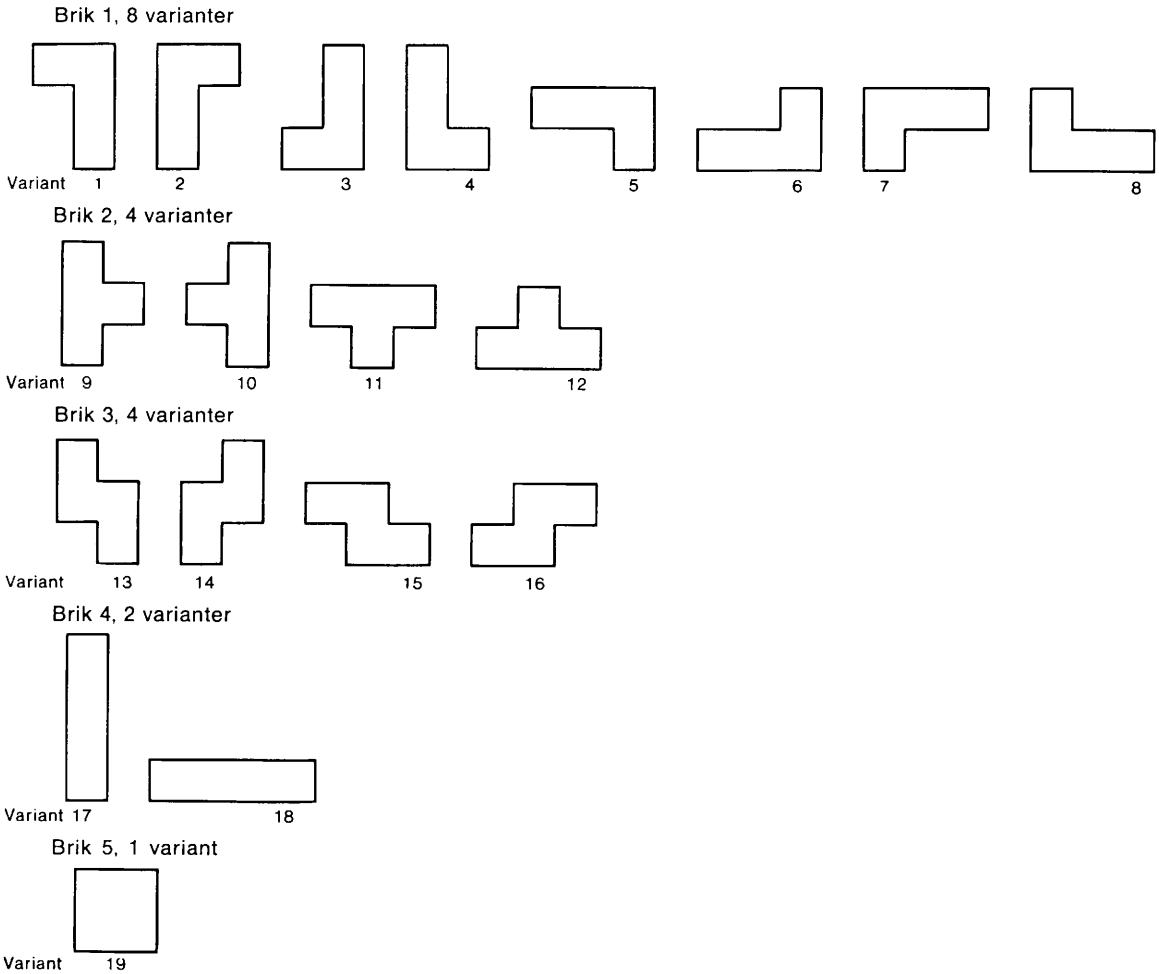




De 63 varianter af pentomino-brikkerne

Det første skridt i systematiseringen er at vi opstiller alle varianter af brikkerne, dvs. de forskellige figurer der fremkommer ved at man vender og drejer hver brik på alle måder der kan optræde når brikken anbringes på brættet. Vi når da frem til følgende opstilling, der viser de fem brikker og deres i alt 19 varianter:

Tetromino brikvarianter



Formålet med at opstille varianterne på denne systematiske måde er at det bliver muligt at gå dem igennem, én efter én, og være sikker på at alle tænkelige muligheder er udtømt når vi er nået til den sidste. Vi skal blot altid tage brikvarianterne i en bestemt rækkefølge, f. eks. i netop den rækkefølge der er vist i opstillingen.

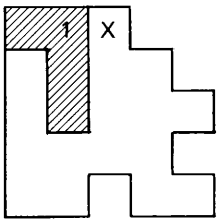
Det andet skridt i systematiseringen er at vedtage en lignende systematisk rækkefølge for brættets felter, som skal være den rækkefølge i hvilken vi vil forsøge at dække felterne med brikker. Vi vil vedtage at gå frem på samme måde som ved læsning, altså begynde øverst til venstre og gå frem mod højre, linie for linie.

For det tredje vedtager vi at fylde brættet op på den måde at vi altid forsøger at få det næste fri felt dækket med den næste variant for en brik der hidtil ikke har været prøvet i denne sammenhæng. Hvis varianten passer går vi videre med at forsøge det næste fri felt. Når det viser sig at være umuligt at komme videre skal vi fjerne den sidste variant der er blevet pla-

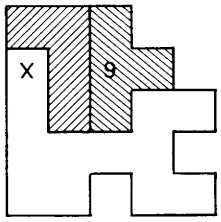
ceret og fortsætte med den næste variant i rækkefølge.

For det fjerde vil vi ikke blive stående ved den første løsning vi finder, men efter at have noteret den på passende måde gå videre som om den sidst placerede variant ikke passede. På den måde vil vi finde alle løsninger, hvis der eksisterer nogen.

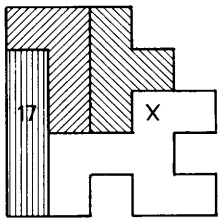
Hvis vi begynder forfra på denne måde får vi hurtigt lagt fire brikker på følgende måde:



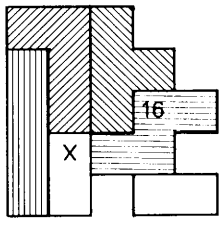
Stilling 1



Stilling 2



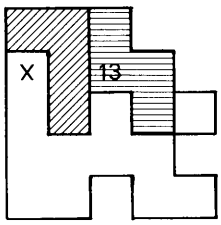
Stilling 3



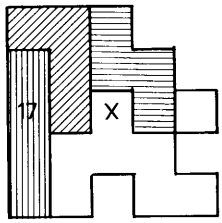
Stilling 4

Den sidst lagte brik er markeret med brikvariantnummeret. Det første fri felt der skal dækkes i næste forsøg markeres med kryds. Placeringen af de to første brikker går helt af sig selv. I stilling 2 undersøger vi først de fire varianter af brik 3, men finder at ingen af disse kan placeres til at dække krydset uden at dække andre brikker eller nå uden for brættet. Vi fortsætter derfor med brikvariant 17 og finder at den kan bruges. I stilling 3 begynder vi med den første hidtil ubrugte brik, nr. 3. Dens varianter 13, 14 og 15 må forkastes, mens variant 16 kan placeres, stilling 4. Nu er kun brik 5 tilbage, som imidlertid ikke kan placeres.

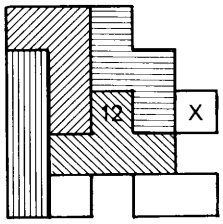
Da vi ikke kan komme videre i stilling 4 må vi fjerne den sidst placerede variant, 16, og vender således tilbage til stilling 3. Her skal vi stadig dække krydset, men vi skal bruge en variant der følger efter 16. Af sådanne findes kun brik 5, som ikke kan bruges i stilling 3. Vi kan altså slet ikke komme videre fra stilling 3 og må gå tilbage til stilling 2, idet vi skal fortsætte efter brikvariant 17. Da imidlertid hverken variant 18 eller 19 kan bruges er også stilling 2 ubrugelig og vi må gå helt tilbage til stilling 1 og fortsætte fra variant 9. Vi finder da følgende tre nye stillinger:



Stilling 5

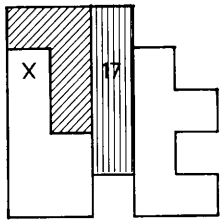


Stilling 6



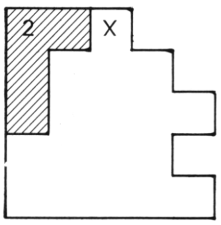
Stilling 7

I stilling 7 går vi igen i stå og det viser sig at heller ikke stilling 6 og 5 tillader andre veje frem, og vi er da igen tilbage i stilling 1. Denne tillader endnu en enkelt udvikling:



Stilling 8

Herfra er alle veje dog igen lukket, og vi når da til at selve stilling 1 er umulig, og vi må gå helt tilbage til det tomme bræt og forsøge med den næste variant i øverste venstre hjørne:

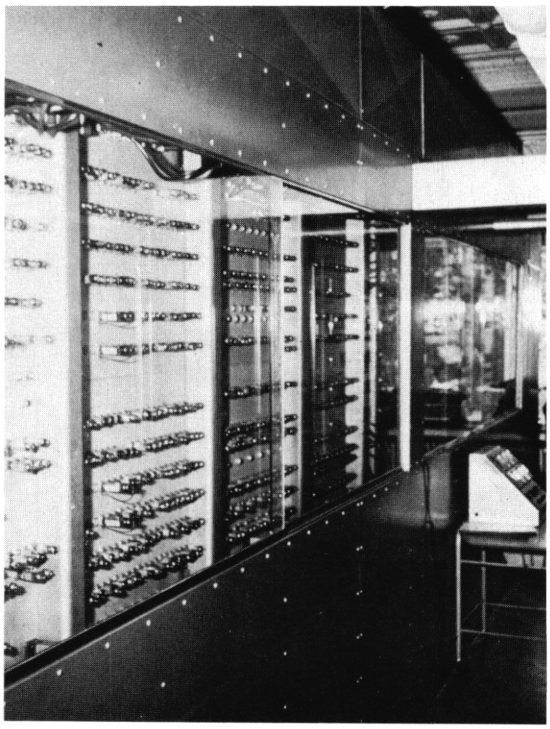


Stilling 9

Dette viser sig at være en god begyndelse – prøv selv at finde den rigtige løsning ud fra denne begyndelse! Der findes i øvrigt endnu en løsning, der begynder på en anden måde. Prøv også at finde den!

Forhåbentlig er det nu klart at den metode der er beskrevet er ganske systematisk. På ethvert trin er det der skal gøres givet gennem reglerne for rækkefølgen af varianterne og felterne. Hvad der måske ikke er helt klart er at vi med den givne fremgangsmåde vil finde *alle* forskellige løsninger, hvis der overhovedet er nogen. Dette er dog ikke alt for svært at indse. Lad os tænke på en bestemt færdig løsning. I sit øverste venstre hjørne må den have en af varianterne. I det næste fri felt af brættet må den nødvendigvis have en anden brik i en af *dens* varianter, og så fremdeles. Da vi med vor systematiske fremgangsmåde forsøger at anbringe enhver variant i det øverste venstre hjørne, og dernæst enhver ubrugt brik i enhver af dens varianter i næste fri felt, osv., da må vi nødvendigvis nå frem også til denne bestemte løsning.

Vi er nu igennem første halvdel af beskrivelsen af hvordan puslespilopgaven kan løses. Vi har beskrevet en systematisk fremgangsmåde, som må kunne udføres rent automatisk. Tilbage står den anden halvdel, at beskrive hvordan denne fremgangsmåde kan realiseres med de midler der er til rådighed i en datamat. Som

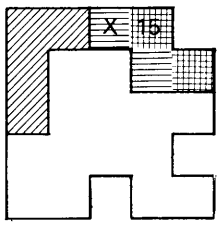


I 1955 påbegyndtes bygningen af DASK, den første elektroniske datamat i Danmark. Den blev bygget som en noget ændret udgave af den svenske maskine BESK af en lille gruppe teknikere på Regnecentralen i Valby under ledelse af civilingeniør **B. Scharøe Petersen.** DASK var køreklar i efteråret 1957 men blev sidenhen udbygget i flere omgange dels for at forøge lagerkapaciteten, dels for at tilslutte nye kontaktorganer. I 1967 – efter kun 10 års forløb – blev DASK demonteret fordi den var overgået af de nyere, mere avancerede datamater, men store dele af den er bevaret og findes nu på Teknisk Museum i Helsingør.
Blandt de mange højst forskelligartede opgaver, som DASK har løst i løbet af de 10 år, skal nævnes valg-beregningerne i forbindelse med TV-udsendelsen ved folketingsvalget i 1960 – første gang fjernsynet herhjemme benyttede en datamat.

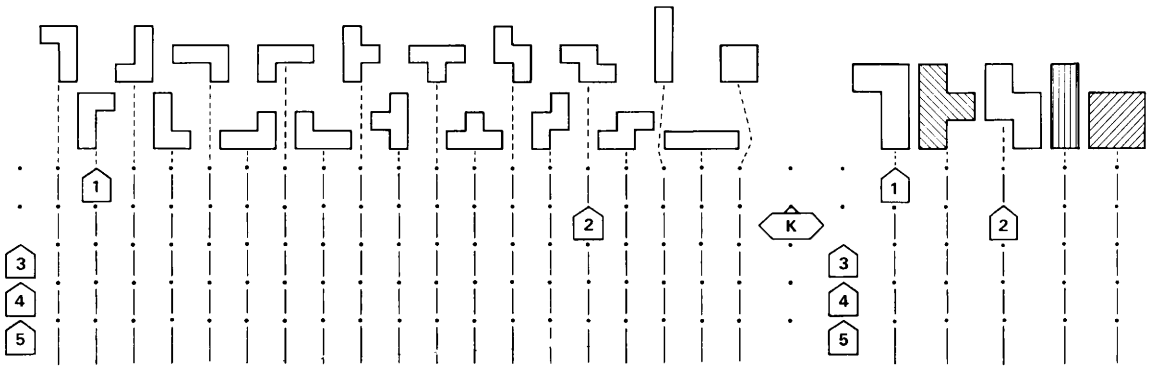
vi hidtil har beskrevet fremgangsmåden kan den ikke direkte bruges, dels fordi de regler vi følger for at forsøge placeringer af brikkerne kun er udtrykt i ord, dels fordi der er tale om at flytte rundt med brikker og at se om de passer inden for brættets felter, ting som ikke kan udføres af en datamat. Vi har endnu tilbage at repræsentere disse forhold med passende data, som tillader at en datamat kan gen-

nemføre de forskellige forsøg med placeringer af brikkerne.

Lad os først overveje hvordan vi kan udtrykke reglerne for de forsøgsvise placeringer af brikkerne. Disse regler fortæller os hvilken variant der skal prøves næste gang når brikkernes stilling er givet, og afhænger således af at vi kan beskrive en bestemt forsøgsstilling. For at nå frem til en beskrivelse helt ved hjælp af data vil vi benytte et mellemled, en beskrivelse der består af rigtige brikker og nogle flytbare pegepinde eller pile. For at forklare den vil vi vise hvordan den ser ud når vi forsøger at placere brikvariant 15 i stilling 9:



Vi beskriver de brikker og varianter der er benyttet gennem følgende opstilling:



I opstillingens højre halvdel holder vi styr på brikkerne selv. De der ikke er brugt er skraveret. De der for øjeblikket er placeret på brættet markeres med pilene betegnet 1 til 5, idet den der er placeret først markeres af pil 1. Disse fem brikpile kan forskydes langs vandrette linier. Pilen K i midten peger på den linie der bærer brikpilen hørende til den sidst placerede brik. Den pil, som K peger på, vil vi også betegne som *den aktuelle brikpil*. Den peger på den aktuelle brik.

I opstillingens venstre halvdel holder vi regnskab med brikkernes varianter. De fem variantpile, 1 til 5, svarer ganske til brikpilene. Den variantpil som K peger på betegner vi *den aktuelle variantpil*. Den peger selv på den aktuelle variant.

Vi kan nu udtrykke reglerne for hvilken placering der næste gang skal forsøges, ved hjælp af pilene. Pilene beskriver en situation hvor de varianter de udpeger, på nær den aktuelle, har kunnet placeres på brættet. Hvad den aktuelle brikvariant angår, er vi netop ved at forsøge at placere den i den første fri position på brættet. Der er da to muligheder, enten at den aktuelle variant ikke kan placeres på brættet, eller at den kan placeres. Det eksempel der er brugt ovenfor viser et tilfælde hvor varianten ikke kan placeres, da den rager uden for brættet. Vi udtrykker det der skal gøres som et program, skrevet i sædvanligt sprog.

Program for brikvalg

Del 1, den aktuelle variant passer ikke på brættet.

Passer ikke:

Den aktuelle variantpil forsøges rykket til næste variant mod højre. Hvis der findes flere varianter af den samme brik forsøges med den næste variant. I modsat fald lægges den aktuelle brik tilbage på plads blandt de ubrugte brikker, og vi fortsætter som beskrevet herefter.

Ny brik:

Den aktuelle brikpil forsøges rykket til højre. Hvis der findes flere ubrugte brikker til højre rykkes den aktuelle brikpil hen til den første af disse og den aktuelle variantpil rykkes til den første af dens varianter, som nu søges placeret på brættet. I modsat fald rykkes både den aktuelle brikpil og den aktuelle variantpil helt til venstre, K-pilen rykkes et trin opad, hvorved den foregående brik og variant bliver aktuel. Herefter udføres den samme programdel som skulle have været udført såfremt den nu aktuelle variant ikke passede, dvs. vi fortsætter fra det sted ovenfor der betegnes »Passer ikke«.

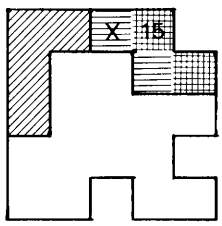
Del 2, den aktuelle variant passer på brættet.

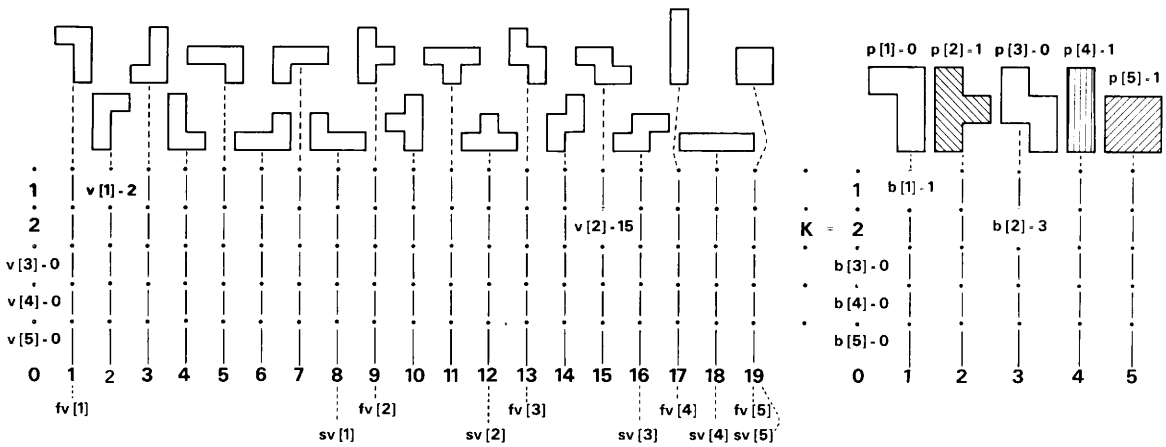
Variant passer:

Såfremt K-pilen peger på den sidste brik har vi fundet en løsning, som vi kan notere os. Da skal vi fortsætte med søgningen efter flere løsninger ved at gå videre ovenfor fra »Passer ikke». I alle andre tilfælde rykker vi K-pilen et trin ned og med den tilsvarende nye, aktuelle brikpil fortsætter vi ovenfor fra stedet »Ny brik«.

Vi har hermed gennemført en beskrivelse af brikvalget, som benytter sig af egentlige brikker og bevægelige pile. Det yderligere skridt der bringer os herfra til en rent datamæssig beskrivelse kræver visse detaljer yderligere udpenslet, men bringer intet principelt nyt.

Den rent datamæssige beskrivelse kræver først at vi erstatter hele stillingsbeskrivelsen, med dens brikker og pile, med variable hvis værdier gør det ud for pilens stilling, m. m. For at vise hvordan dette kan gøres viser vi igen det samme eksempel, men med en række variable tilføjet.





Øverst viser vi igen brættet med de to brikker der er forsøgt placeret. Den del af den anden brik der rækker ud over brættet er vist dobbelt skraveret. Til højre holder de fem variable, p [1] til p [5], regnskab med hvilke brikker der for øjeblikket er placeret, svarende til skraveringen. Brikkerne selv repræsenteres ved tallene 1, 2, . . ., 5. Til at repræsentere de fem brikpile har vi de fem variable, b [1] til b [5]. Ved deres rækkefølge fortæller de i hvilken orden brikkerne er brugt. K-pilen repræsenteres ved en variabel, K, med en værdi mellem 1 og 5. Varianterne nummereres som sædvanlig fra 1 til 19. Til at vise hvilke af disse 19 varianter der hører til hver brik har vi fem variable, fv [1] til fv [5], der angiver den

første variant for hver brik, og andre fem variable, sv [1] til sv [5], der angiver den sidste variant. Endelig angiver fem variable, v [1] til v [5], hvilke varianter der er placeret på brættet.

Af opstillingen fremgår:

Nummeret for den aktuelle brik:	b [K]
Nummeret for den aktuelle variant:	v [K]
Første variant for den aktuelle brik:	fv [b[K]]
Sidste variant for den aktuelle brik:	sv [b[K]]

Det springende punkt er at forstå disse indicerede variable rigtigt. Når der står v [K] menes der den af de fem variable v [1] til v [5] der udpeges af den øjeblikkelige værdi for K. K ændrer sig under programmets forløb og man skal til enhver tid bruge dens øjeblikkelige værdi. Tilsvarende gælder ved et udtryk som sv [b[K]]. Her skal man tage den øjeblikkelige værdi for K og med den udvælge en af de fem variable b [1] til b [5]. Værdien af denne b-variabel, b [K], som er nummeret på den aktuelle brik, skal derefter bruges til at udpege en af de fem variable sv [1] til sv [5].

Som yderligere forberedelse til at opskrive brikvalget som et Algol program må det nævnes at dette vil henvise til andre programdele, som ikke er medtaget:

Prøv variant: her begynder en programdel som forsøger at placere den brikvariant der udpeges af v [K] i næste fri felt på brættet. Denne programdel vil fortsætte, enten ved »Passer ikke« eller ved »Variant passer«, afhængig af resultatet af dens arbejde. Nedenfor skal vi se nøjere på hvordan programdelen »Prøv variant« kan udformes som et Algol-program.

Skriv løsning ud: denne programdel sættes i arbejde når brættet er fyldt. De varianter der er placeret er angivet i v [1] til v [5]. Vi vil ikke komme ind på hvordan den færdige løsning afleveres fra programmet.

Søgning færdig: her er alle muligheder undersøgt. Hvis der undervejs er fundet løsninger er de allerede afleveret, som nævnt ovenfor. Programmet har derfor kørt til ende.

Det følgende Algol program svarer ganske nøje til de regler der er givet ovenfor i »Program for brikvalg«.

Algolprogram for brikvalg

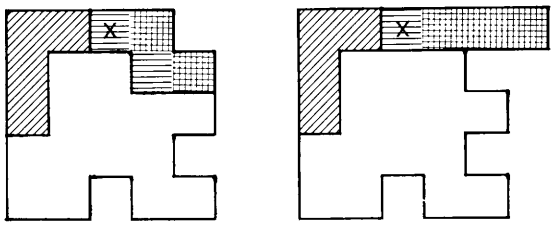
```
Passer ikke:
 1. if v[K] ≠ sv[b[K]] then
 2.     begin v[K] := v[K] + 1;
 3.         go to Prøv variant;
 4.     end;
 5. p[b[K]] := 1; comment Den aktuelle brik lægges tilbage på plads blandt de
                  ubrugte brikker;
 6. Ny brik:
 7. if b[K] < 5 then
 8.     begin b[K] := b[K] + 1;
 9.         if p[b[K]] = 0 then go to Ny brik;
10.         p[b[K]] := 0;
11.         v[K] := fv[b[K]];
12.         go to Prøv variant;
13.     end;
14.     if K = 1 then go to Søgning færdig;
15.     b[K] := 0;
16.     v[K] := 0;
17.     K := K - 1;
18.     go to Passer ikke;

Variant passer:
19.     if K = 5 then
20.     begin Skriv løsning ud;
21.         go to Passer ikke;
22.     end;
23.     K := K + 1;
24.     go to Ny brik;
```

For at forstå dette program bør man for det første sammenligne det i enkeltheder med det tilsvarende »Program for brikvalg« udtrykt i sædvanlig tekst.

For det andet skal man »lege maskine«, dvs. følge programmets forløb i et konkret tilfælde, ganske som en maskine ville gøre det. Hertil må man holde nøje regnskab med alle variables øjeblikkelige værdier, sådan som de er på det stadium man er nået til. Dette gøres bedst ved at man laver en tabel hvor hver variabel har en kolonne, hvori dens værdi kan opskrives. Hver linie repræsenterer et stadium af forløbet. Ved at følge linierne kan man altså i kolonnerne se hvilke variable der har fået ændret værdi. I kolonnen til venstre anføres numrene på de linier i programmet der er udført på hvert stadium. Værdierne for de variable skrives kun når de ændres. På denne måde kan vi følge udviklingen fra den stilling vi har brugt som illustration ovenfor, A, til en ny stil-

ling, B, der blot er fremkommet ved at vi har brudt af et tilfældigt sted i forløbet.



Stilling A Stilling B

Aktiv del af Algol program, linie nummer	K	p[ ] 1 2 3 4 5	b[ ] 1 2 3 4 5	v[ ] 1 2 3 4 5
Stilling A	2	0 1 0 1 1	1 3 0 0 0	2 15 0 0 0
Passer ikke, 1, 2, 3				16
Passer ikke, 1, 5		1		
6, 7, 8			4	
9, 10		0		
11, 12				17
Variant passer, 19, 23	3			
24, 6, 7, 8			1	
9, 6, 7, 8			2	
9, 10		0		
11, 12				9
Passer ikke, 1, 2, 3				10
Passer ikke, 1, 2, 3				11
Passer ikke, 1, 2, 3				12
Passer ikke, 1, 5		1		
6, 7, 8			3	
9, 10		0		
11, 12				13
Passer ikke, 1, 2, 3				14
Passer ikke, 1, 2, 3				15
Passer ikke, 1, 2, 3				16
Passer ikke, 1, 5		1		
6, 7, 8			4	
9, 6, 7, 8			5	
9, 10			0	
11, 12				19
Passer ikke, 1, 5			1	
6, 7, 14, 15, 16			0	0
17	2			
18, 1, 2, 3				18
Stilling B	2	0 1 1 0 1	1 4 0 0 0	2 18 0 0 0

Vi skal nu se hvordan man kan indføre en datarepræsentation for brættet og brikkerne, der vil gøre det muligt at få den automatiske proces til også at afgøre hvorvidt en bestemt variant kan placeres i den næste fri position. Til dette formål indfører vi en indiceret variabel, FELT, med et element for hvert felt i brættet, samt nogle ekstra langs kanten, idet felterne nummereres på følgende måde:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Der er altså i alt 36 indicerede variable, FELT[1] til FELT[36]. Disse variable vil blive brugt til at vise hvorvidt det tilsvarende felt er dækket af en brik eller er frit. De felter der ligger uden for brættet, f. eks. 4, 5, 6, vil blive behandlet som om de fra først af var dækket af brikker der ikke kan fjernes. Helt generelt vil vi markere hvert felt således:

FELT[G] = 1 betyder at felt G er dækket eller brugt,

FELT[G] = 0 betyder at felt G er frit.

Til at begynde med, når hele brættet er frit vil værdierne af de 36 variable være:

000111 000011 000001 000011 001001 111111

Grunden til at denne repræsentation af felterne er hensigtsmæssig er at en variant af en brik nu kan repræsenteres ved blot tre tal, nemlig feltnumrene for tre af variantens felter regnet i forhold til numret for hovedfeltet, dvs. feltet længst til venstre i første linie. Tag for eksempel en placering af variant 3 af brik 1:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Hovedfeltet for brikken er placeret i felt nummer 9. De tre øvrige felter er 15, 20 og 21, som i forhold til hovedfeltet er 6, 11 og 12. Anbringer vi samme variant med hovedfeltet i nummer 17 får vi de tre øvrige felter ved at lægge 17 til de tre tal, 6, 11 og 12: 17 + 6 = 23, 17 + 11 = 28 og 17 + 12 = 29. Vi ser således at denne variant fuldstændigt kan beskrives med de tre tal, 6, 11, 12. Det samme gælder alle de øvrige varianter, og hele sættet af varianter kan beskrives med en tabel:

Variant	Relative feltnumre		
1	1	7	13
2	1	6	12
3	6	11	12
4	6	12	13
5	1	2	8
6	4	5	6
7	1	2	6
8	6	7	8
9	6	7	12
10	5	6	12
11	1	2	7
12	5	6	7
13	6	7	13
14	5	6	11
15	1	7	8
16	1	5	6
17	6	12	18
18	1	2	3
19	1	6	7

Vi vil benytte tre sæt af indicerede variable, ANDEN, TREDJE og FJERDE, til at opbevare disse tal. Da har vi for eksempel for variant 17: ANDEN[17]= 6, TREDJE[17] = 12 og FJERDE[17] = 18. For at afgøre hvorvidt variant V kan placeres med sit hovedfelt i felt nummer F, idet vi går ud fra at feltet F selv er frit, må vi prøve de tre øvrige felter et efter et:

```
if FELT [F + ANDEN [V]] = 1 then go to Nej;
if FELT [F + TREDJE [V]] = 1 then go to Nej;
if FELT [F + FJERDE [V]] = 1 then go to Nej;
```

Her fortsættes hvis varianten kan placeres. Dette er den centrale programdel der konstaterer om en opgivet variant kan placeres på brættet eller ej.

På ganske tilsvarende måde kan vi opskrive de ordrer der placerer variant V med sit hovedfelt i felt F, idet vi går ud fra at der er fri bane:

```
FELT [F] := 1;
FELT [F + ANDEN [V]] := 1;
FELT [F + TREDJE [V]] := 1;
FELT [F + FJERDE [V]] := 1;
```

Ordrerne der fjerner en opgivet variant fra en opgivet placering er ganske mage til, bortset fra at tallet 1 skal erstattes med 0.

For at finde det første fri felt, der står over for at dækkes, skal vi løbe felterne igennem forfra indtil vi finder et frit:

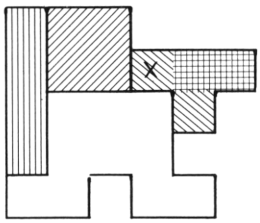
```
    FF := 0;
Søg første fri: FF := FF + 1;
    if FELT [FF] = 1 then go to Søg første fri;
```

For at kunne placere og fjerne varianterne må vi yderligere holde regnskab med hvilke varianter der til enhver tid er placeret hvor på brættet. Vi ved at der højst er placeret fem varianter, og det er derfor tilstrækkeligt at benytte fem indicerede variable, VAR [1] til VAR [5], som angiver numrene på de varianter der for øjeblikket er placeret, og andre fem variable, FØRSTE [1] til FØRSTE [5], der angiver feltnumrene for disse varianter. Hertil har vi brug for en enkelt variabel, P, der angiver hvormange varianter der for øjeblikket er placeret. Værdien af P vil ofte være den samme som værdien af K, men på grund af den adskillelse vi har gjort mellem programmet til brikvalg og den del der holder regnskab med brættet, er overensstemmelsen dog ikke fuldstændig.

Hvad der her er gennemgået viser væsentlige dele af et program til løsning af puslespilopgaver, helt ned til de detaljer som datamaterne kræver. Det der er udeladt er dels en programdel der forbereder søgningen efter løsninger på den måde det er beskrevet, ud fra en beskrivelse af brættet og brikkerne, dels den del der afleverer resultaterne, og endelig den samlede struktur og styring som får de enkelte dele til at virke rigtigt sammen.

Opgave 6.1.

Opskriv værdierne af alle variable K, p [1] til p [5], b [1] til b [5] og v [1] til v [5] som de forefindes når søgningen efter en løsning til puslespillet er nået til følgende stilling:



Følg derefter programmet til udvælgelse af næste variant skridt for skridt og hold regnskab med alle variable indtil der findes en løsning.

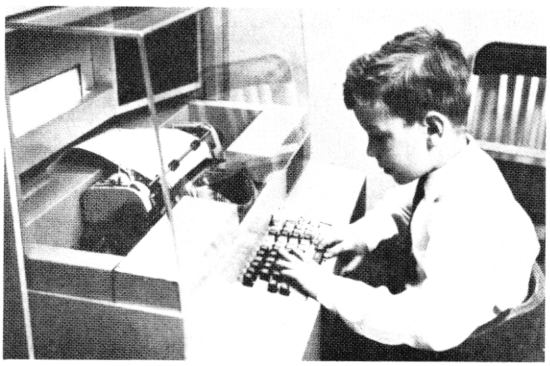
John von Neumann (1903–57) som i mange år arbejdede ved Princeton Institute of Advanced Study i USA var en af vor tids betydeligste matematikere; samtidig havde han stor interesse for matematikkens anvendelse inden for mange felter som f. eks. kvantemekanik, hydrodynamik, meteorologi og astronomi, og har ydet væsentlige bidrag til forskningen også inden for disse videnskaber. Han indså meget tidligt betydningen af at få bygget maskiner der kunne udføre meget omfattende beregninger automatisk og hurtigt, og sammen med H. Goldstine udviklede han i 1945–46 de principper hvorefter man kunne bygge en elektronisk datamaskine der blev styret af et lagret program.

7. Datalogien og vort samfund

7.1. Nye anvendelser

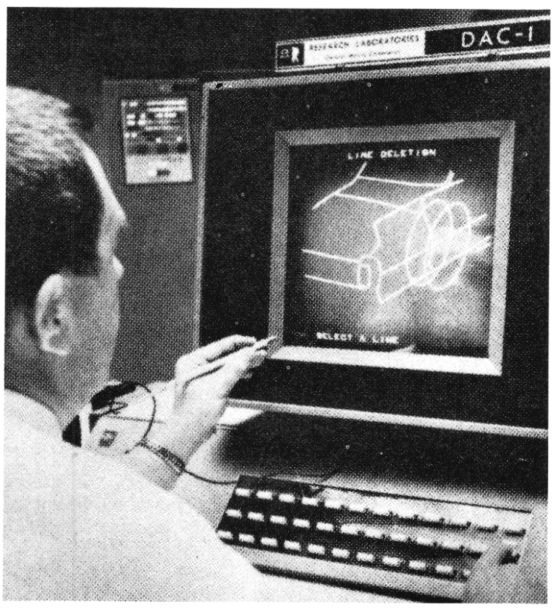
Datamaterne tilpasses stadig mere og mere til de mennesker, der skal bruge dem. Ved hjælp af en *dataskærm,* en slags TV-skærm der er koblet til datamaten, kan man nu arbejde med data i form af tegninger, hvilket åbner mulighed for helt nye anvendelser. På billedet nederst t.v. ses en ingeniør i færd med at konstruere en bil med støtte af en datamat. Med den blyantlignende genstand, som kaldes en *lyspen,* kan han ændre tegningen, og datamaten kan derefter for eks. vise billedet af bilen set fra alle vinkler.

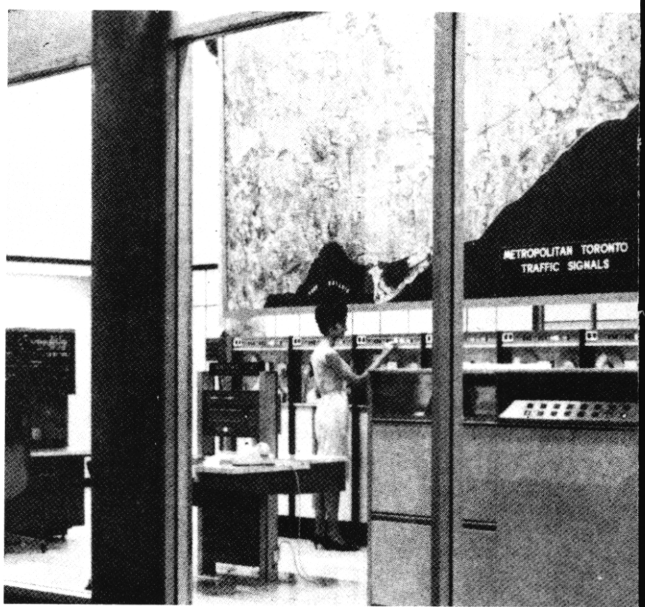
Også inden for undervisning vinder datamaterne frem. Med særlige undervisningsprogrammer, der på en skrivemaskine stiller spørgsmål til eleven og vejleder ham på forskellig måde, er det muligt at lære børn f. eks. at regne uden



hjælp af menneske-lærere. Ved at kombinere skrivemaskinen med en båndoptager der også styres af datamaten, kan man også lave programmer til læse- og skriveundervisning. Eleven hører da spørgsmålene og skriver sine svar.

Datamaterne har også vist sig anvendelige ved trafikreguleringen i storbyer. Ved at måle





trafikstrømmen i gaderne i en storby ved hjælp af automatiske tællere, og sende disse data til en datamat, kan man få den til at styre trafiksignalerne efter det øjeblikkelige behov. På billedet nederst t.h. s. 56 ses trafikreguleringscentret på rådhuset i Toronto, Canada.

7.2. Udviklingen i Danmark og udlandet

Den datamatiske udvikling i USA og Vesteuropa er gået meget hurtigt og synes foreløbig at fortsætte med samme tempo. Dette kan man bl. a. illustrere med de kurver og tal vi nu skal gennemgå. Læg mærke til at på flere af figurerne vokser tallene på skalaen meget kraftigt (det er en såkaldt logaritmisk skala) fordi det ellers var umuligt at illustrere den voldsomme udvikling.

Den teknologiske udvikling i de sidste 15 år har gjort de elektroniske dele af datamaterne mindre og mindre: Figur 7.1 viser hvordan det antal enkeltdele (à la relæer), der kan pakkes på 1 cm^3, er vokset og formodentlig stadig vil vokse med tiden takket være den udvikling der i øjeblikket foregår i forskningslaboratorierne.

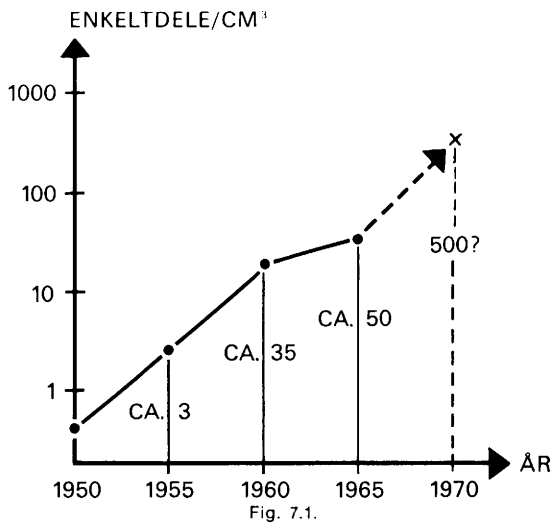


Fig. 7.1.

Dette har været medvirkende til at arbejdshastigheden har kunnet øges mellem 5 og 10 gange hvert femte år: Figur 7.2 viser væksten i det antal simple ordrer (som f. eks. addition af to tal), de nyeste og hurtigste datamater har kunnet udføre på 1 sekund.

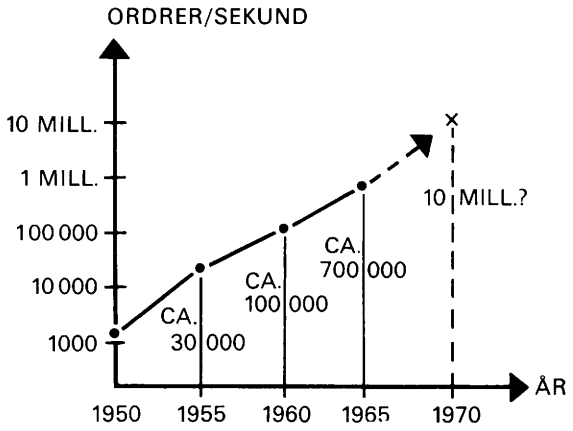


Fig. 7.2.

Denne teknologiske udvikling og det samtidige prisfald på datamater har forårsaget at antallet er vokset med tilsvarende hast, og der er foreløbig ingen tegn til at væksten aftager, dvs. der er ingen tegn på at landene skulle være »mættede« med datamater: Figur 7.3 viser antallet af datamater gennem de sidste 15 år i henholdsvis USA, Vesteuropa og Danmark, samt et groft skøn over hvordan disse antal ventes at udvikle sig i de nærmeste år. (NB: På figuren er skalaen for de danske tal 20 gange så stor som for de øvrige for at man bedre kan sammenligne udviklingstendensen).

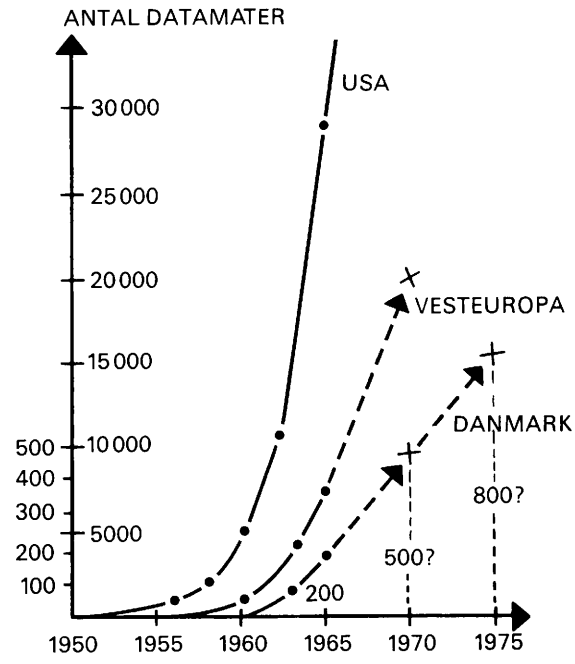


Fig. 7.3.

For at kunne udnytte en datamat fornuftigt regner man med at der skal beskæftiges 10–30 mennesker afhængig af datamatens størrelse, og omtrent halvdelen af disse vil arbejde med planlægning og programmering af de opgaver, datamaten skal løse. Det betyder ifølge figur 7.3 at i 1970 vil alene i Danmark omkring 400 × 20 = 8.000 personer være beskæftiget direkte med datamater, og henved halvdelen af dem med programmerings- og planlægningsarbejde; i 1975 må vi regne med at 16–20.000 personer arbejder med datamater.

7.3. Nye uddannelser og erhverv

Arbejdet med datamater kræver adskillige slags personale med forskellige uddannelser.

Hulleassistenterne betjener de hullemaskiner der bruges til at overføre data på skreven form til hulstrimler eller hulkort. Arbejdet kræver en uddannelse i at betjene hullemaskinerne, samt akkuratesse og vedholdenhed.

Datamatoperatørerne udfører den egentlige betjening af datamaterne. De skal sørge for at datamaten fodres med de rigtige hulstrimler og magnetbånd, at skrivemaskinerne er forsynet med papir, og de skal trykke på datamatens knapper, alt sammen efter de instrukser der følger med de programmer der skal køres. Operatørerne må have en uddannelse i at betjene den bestemte datamat de har med at gøre, og bør desuden kende noget til programmering.

Vedligeholdelsesteknikerne skal løbende sørge for at datamaten med al dens udstyr er i perfekt køredygtig stand. De har bestemte daglige tider hvor de kan gennemføre særlige prøvekørsler med datamaten og bliver i øvrigt tilkaldt hvis der viser sig fejl under den normale brug, og skal da indkredse og udbedre fejlen. Vedligeholdelsesteknikerne må have en uddannelse i elektroteknik og må desuden gennemgå et specialkursus i vedligeholdelsen af den datamat de har ansvaret for. De har brug for evne til at tænke klart og systematisk.

Programmørerne skriver og afprøver de programmer som styrer datamaterne, og må tillige udarbejde beskrivelser af disse programmer, bl. a. til brug for datamatoperatørerne. Der vil normalt kræves mindst matematisk studentereksamen med efterfølgende kursus i programmering. Programmørerne må have evne til at udtænke dataprocesser og til at bevare overblikket over de ofte ret komplicerede programmer de skal udarbejde.

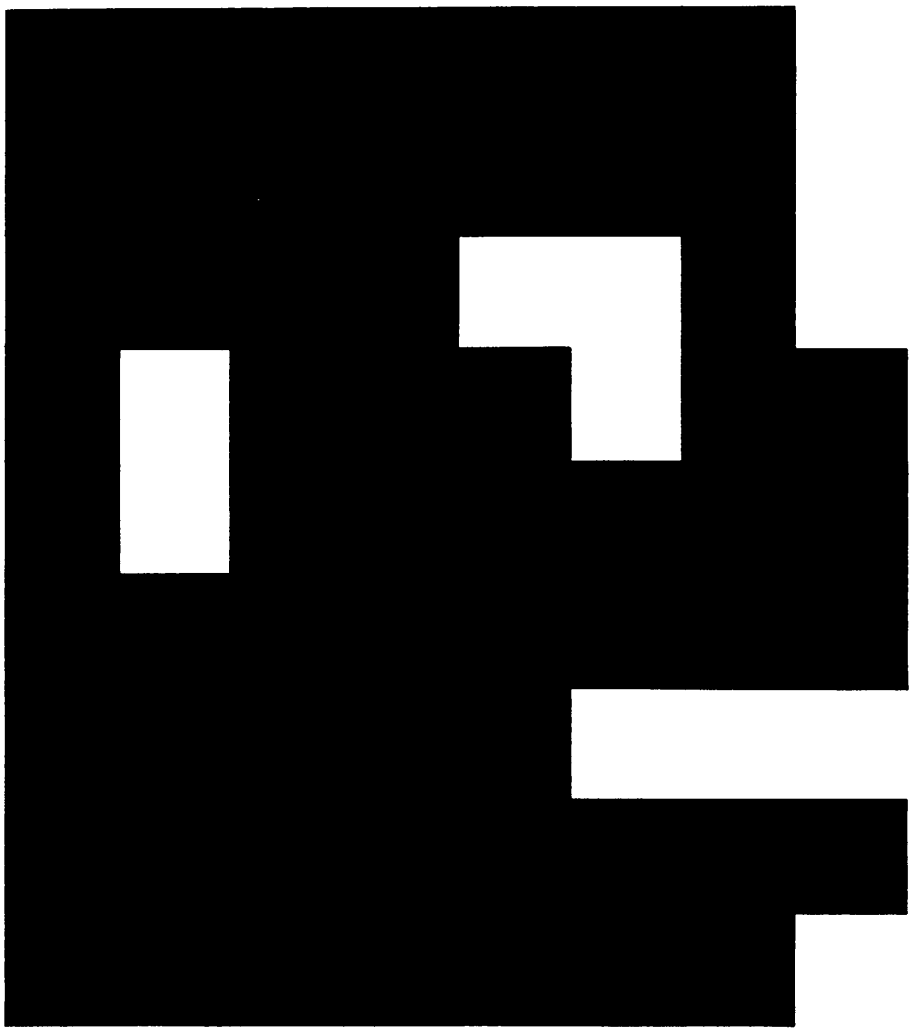
Datamatikerne, også betegnet EDB-konsulenter eller EDB-planlæggere, (EDB står for Elektronisk DataBehandling), har til opgave at tilrettelægge de opgaver som foreligger i det virkelige liv, i fabrikkerne, i kontorerne, i statens administration, i forsvaret, i laboratorierne, osv. på en sådan måde at programmørerne kan gå i gang med at udarbejde selve de programmer der får datamaten til at løse opgaverne. Datamatikerne skal finde ud af hvorledes de nødvendige data bringes til veje, hvorledes disse skal behandles datamatisk, hvilke resultater der skal fremkomme, og hvilke roller de forskellige personer der medvirker skal have. Det kræves normalt at datamatikere har en højere uddannelse, f. eks. som ingeniør, økonom, eller forsker, og dertil at de er fuldt fortrolige med programmering og med datamaternes opbygning og virkemåde.

Uddannelsen til disse forskellige stillinger er stadig i støbeskeen. Megen uddannelse foregår ved datamatfabrikantfirmaerne og de såkaldte datamat- eller EDB-service firmaer, men der findes også særlige EDB-skoler som giver kurser i disse fag. Endelig sker der en hastig udvikling i undervisningsaktiviteten ved universiteterne, de højere læreanstalter, de tekniske skoler, seminarierne og det almene skolevæsen.

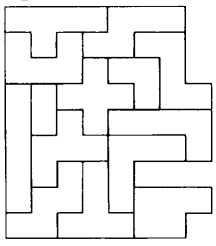
Pentomino puslespil

Til illustration af hvorledes man angriber en ikke helt simpel opgave med brug af en datamat benyttes en puslespil-opgave, der går ud på at dække det nedenfor viste bræt med de tolv såkaldte pentomino-brikker, der også vises nedenfor. Kun de sorte dele af brættet må dækkes.

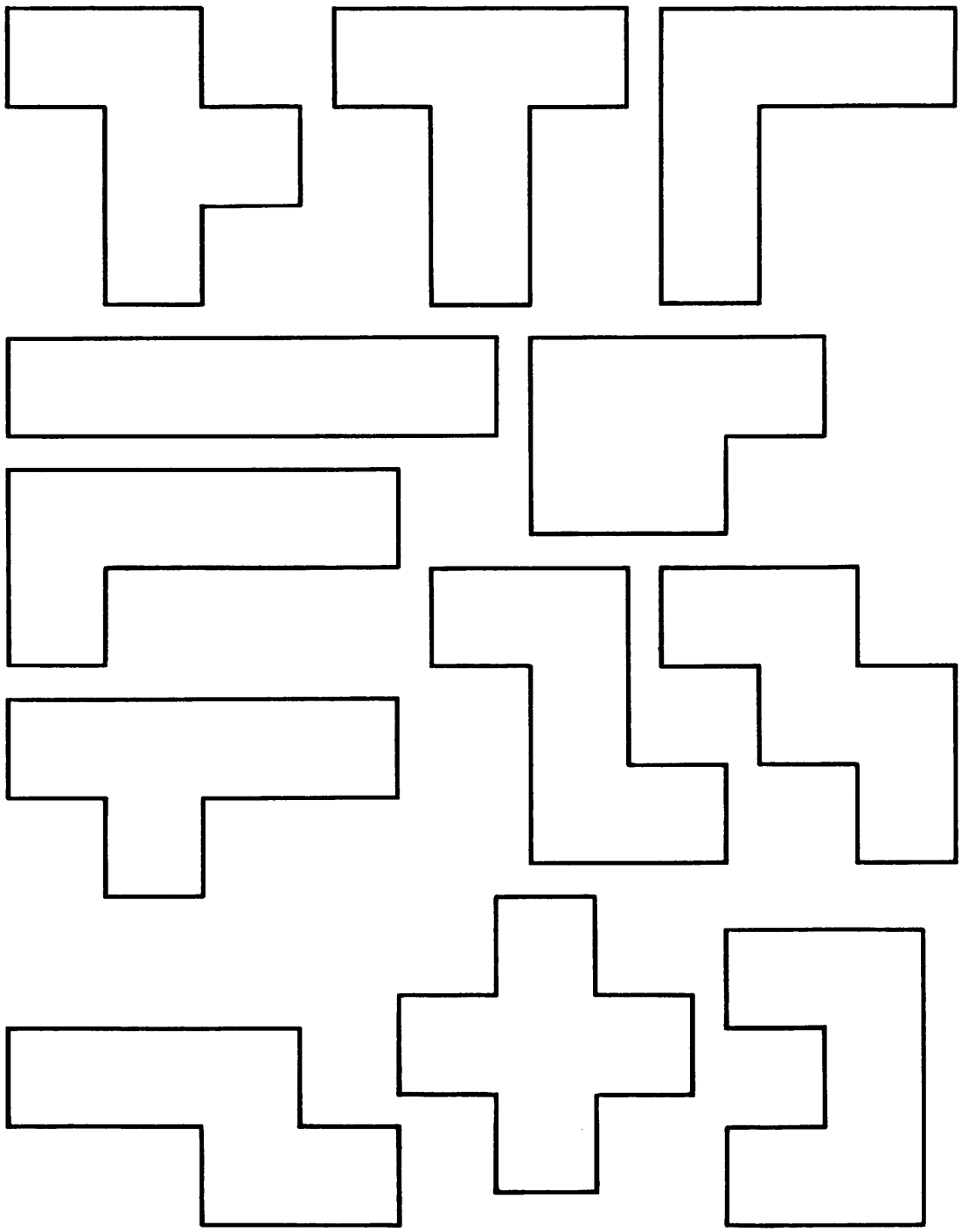
En pentomino består af 5 lige store kvadrater, ligesom en domino består af 2 kvadrater. I den 6. udsendelse arbejdes der også med et tetromino-puslespil, hvor brikkerne hver har 4 kvadrater.



Et eksempel
på en løsning:



De tolv pentomino-brikker. Brikkerne findes også trykt på et løst ark karton, så de kan klippes ud.



INDEX