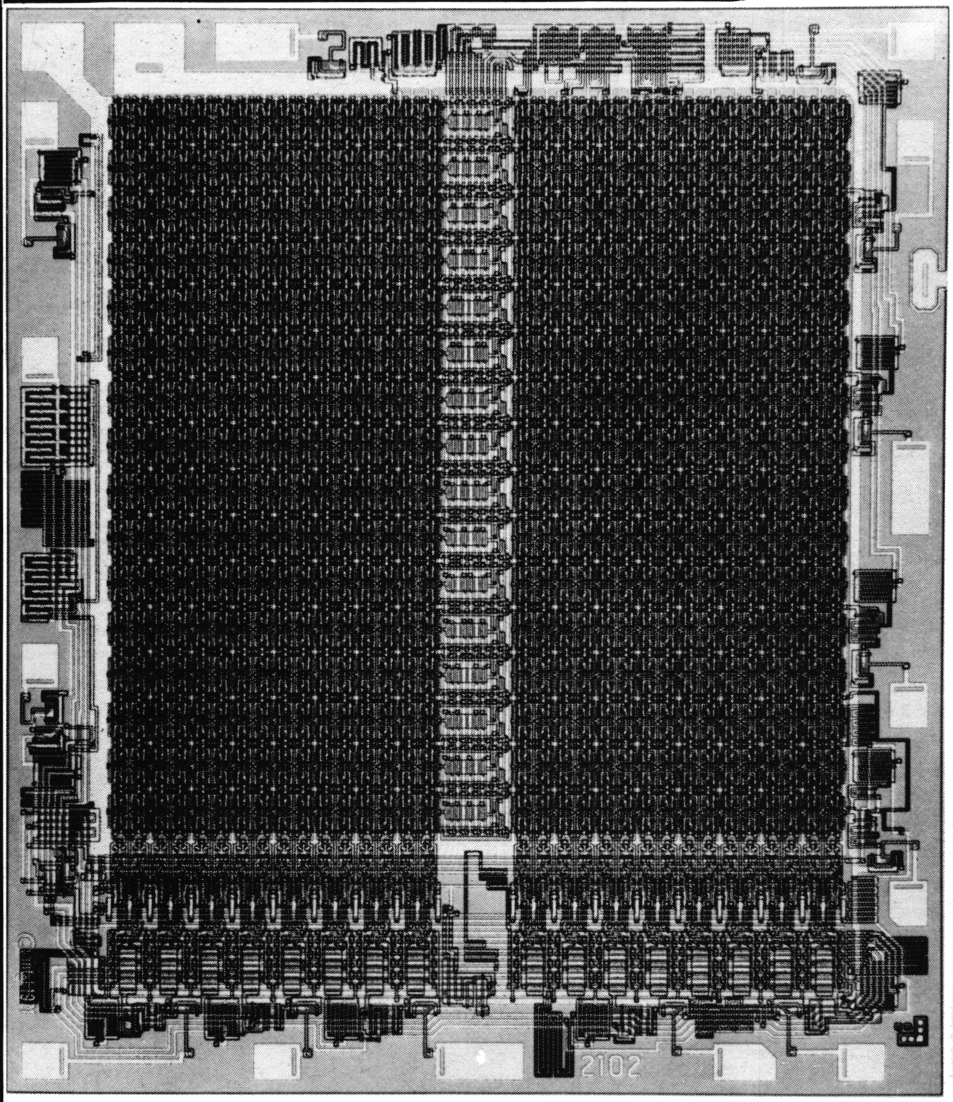
11 Håndbog for datamat-amatører

8 BIT

1977



INDHOLDSFORTEGNELSE

FORSIDEN. Sidste måned viste vi et foto i forstørrelse af en dynamisk 1024 BIT RAM. Denne måned - i samme grad af forstørrelse - afbilleder vi en statisk RAM. (Intel 2102). Også her har vi 1024 BIT, og det kan straks ses ved sammenligning med sidste måneds forside, at der kræves adskilligt flere komponenter til at skabe et statisk RAM i sammenligning med et dynamisk RAM.

ABONNENTER

Vi har allerede nu omkring 1.200 abonnenter på **BIT**. Det lyder måske ikke af meget, men vi er godt tilfredse med dette antal, for indholdet er jo temmelig specielt.

Vi vil naturligvis gerne have endnu flere læsere, for det vil sætte os i stand til at gøre bladet større. Fortæl derfor venner og bekendte om **BIT**.

Kommende numre vil blive trykt i oplag, som er justeret efter den forhåbentlige fortsatte stigning i abonnenter, men det kan vi selvfølgelig ikke gøre med de numre, som allerede er produceret.

Nr. 9 og nr. 10. ligger idag på redaktionen i et oplag på ca. 800 stk. hver, og det er altså det største antal abonnenter, som vil kunne få hele bladet lige fra starten.

NAVNEFORANDRING

Der er ingen vej uden om - **BIT** skal tage navneforandring. Vi er dog så heldige, at vi har fået en meget fleksibel tidsfrist, som vi naturligvis vil benytte til både at finde det helt rigtige navn og til at sikre os, at vi ikke derved interferer med andre interesser.

Flere læsere har været så venlige at give os flere ideer, og vi vil gerne opfordre alle til at hjælpe os.

I næste nummer af **BIT** (det hedder det trods alt endnu) vil vi lave en rask lille konkurrence, hvor vi vil bede læserne stemme på de forslag, vi da bringer.

Da vi gerne vil give den, der kommer med det bedste forslag, en lille præmie, må vi bede jer om at benytte brevkortet bagest i bladet, så vi har et navn og en adresse i forbindelse med forslagene. Vi har ikke nedskrevet navne i forbindelse med indtelefonerede forslag, så de, der allerede har givet ideer ad den vej, må desværre gentage spøgen via brevkortet, hvis de vil sikre sig en chance for at deltage i præmieuddelingen.

Navnet bør være kort, helst 3 eller 4 bogstaver, og det skal naturligvis have forbindelse med indholdet af bladet.

Ordet må være let at udtale, og det bør helst ikke kunne forveksles med andre begreber af anden betydning.

Dette gør naturligvis det hele meget sværere, men vi stoler alligevel på, at vi får en mængde gode forslag.

BIT udgives i løsbladsformat af Telepress ApS.
Adresse:
Greve strandvej 42
2670 Greve Strand
Telefon (02) 90 86 00

Ansvarshavende udgiver
H. Lind
Chefredaktør
Peter Holm
Annoncer
H. G. Lind
Layout
Telepress Aps
Tryk
Rounborg, Skive

Abonnement
11 fortløbende numre/ 1 årgang
kr. 75,00 incl. porto
i Danmark, Sverige og Norge.
Postgiro
1 15 53 69

Bladet modtager gerne manuskripter, men påtager sig intet ansvar for materiale, som tilsendes uopfordret.
Retur garanteret ved frankeret svarkuvert.
Alle henvendelser skriftligt eller i ekspeditionstiden mandag - fredag kl. 9 - 15.

BIT udkommer den 1. torsdag i måneden - juli undtaget
Det først udsendte nummer er nr. 9/1977.
Enkelte numre kan rekvireres fra redaktionen mod fremsendelse af kr. 10,- (incl. porto og eksp.)
Ved abonnementsbestillinger anføres det nummer, som først ønskes leveret.

Tal-systemer

Datamat-amatører kommer til at arbejde med ialt 4 talsystemer. Vi bruger til daglig 10-tal systemet, mens datamaterne kræver vort kendskab til også 2-tal, 8-tal og 16-tal systemet. Heldigvis er dette ikke så svært at beherske, som det umiddelbart kan synes - blot man går lidt systematisk til værks. Og det gør vi her.

Det talsystem, som vi beskæftiger os med til daglig, benævnes det decimale tal-system. Vi er vant til at tælle og regne med 10, og derfor er det vanskeligt for os at arbejde med andre talsystemer, selvom ethvert ciffer - større end 1 - kan være basis for et talsystem.
I datamater arbejder vi med binære tal i den interne maskinkode, mens vi i den maskinkode, som vi nedskriver, arbejder med oktal- eller hexadecimalsystemet. For de flæste er det frygtelig forvirrende at benytte flere talsystemer ad gangen, og hvis man udelukkende skal programmere i BASIC eller andet high-level sprog, behøver man ikke at tænke meget over de "skæve" talsystemer.
Det er dog praktisk at have et grundliggende kendskab til de interne forhold i datamaterne, og dertil hører desværre disse andre talsystemer.

DANNELSE AF ET TALSYSTEM

Det skyldes formodentlig, at vi har 10 fingre, at 10-tal systemet anvendes overalt idag. Ret betragtet er der dog intet magisk ved tallet 10, så lad os for et øjeblik glemme dette absolutte antal og betragte talsystemet udefra, som om vi aldrig havde brugt det før.
Vi bemærker straks, at der ialt findes 10 forskellige symboler: 1, 2, 3, 4, 5, 6, 7, 8, 9 og 0. Da det sidste symbol repræsenterer ingenting, kan vi altså ved brug af kun eet af disse symboler beskrive mængder af 10 forskellige størrelser - inklusive intet.
Da vi selvfølgelig ønsker at kunne omtale endnu større mængder end de, der findes mellem 0 og 9, må vi finde på en måde, hvorpå disse symboler kan sammenstykkes til at udtrykke endnu større mængder.
Den næste højere værdi, som vi skal kunne fremstille, er den, som er 1 mere end 9. Denne konstruerer vi ved sammensætning af 1 og 0, idet 1 denne gang repræsenterer et antal, som afgøres af dets placering.
1-tallet fortæller altså, hvor mange enheder, er 1 større end 9, som findes i den samlede mængde. Denne mængde (som vi kalder 10 i daglig tale), består derfor af een enhed større end 9, og ingen enheder mindre end 9.
Enhederne mindre end 9 kalder vi for enere, og vi skal nu have et godt navn for det antal, som er større end 9 - hvad med TI? En tilsvarende beslutning om navngivning af endnu større mængder giver os HUNDREDER og TUSINDER o.s.v.

Når vi skal beskrive antallet af en større mængde, laver vi i virkeligheden en detaljeret analyse og opstiller tallet, som beskriver mængden, efter følgende procedure:

Antal enere
Antal tiere
Antal hundreder
Antal tusinder
o.s.v.

For at angive hvilke af disse størrelser, det drejer sig om, placerer vi et antal symboler for ingenting, (0) som viser, hvorledes antallet af enheder skal placeres. Hvis der er 2 stk. "0" efter et ciffer, betyder det, at der hverken er enere eller tiere, hvorfor det må dreje sig om hundreder.
Det vil være de fleste bekendt, at vi ikke skriver tallene under hinanden, hvis det drejer sig om en større mængde, som repræsenteres ved både enere, tiere o.s.v. Vi kunne godt skrive et 4-cifret tal på denne måde:

05000
00200
00030
00004

men vi ville hurtigt blive træt af at skrive alle de nuller, som ikke er strengt nødvendige. Vi kan ikke undvære dem, som følger efter tallet (til højre for det), for vi skal angive positionen, men vi kan undvære de, som er til venstre for tallet:

5000
200
30
4

Der er dog stadig en frygtelig mængde cifre at skrive, selvom vi udtaler mængden, som den her er beskrevet, nemlig: *Femtusinde og tohundrede og fire og tredive.* Det er en besynderlig mangel på logik, som i talesproget lader os bytte om på de sidste to enheder, men det er en germansk arv, som vi nok aldrig slipper af med, selvom vi på vore checks skriver det mere rigtigt.
Skarpsindige hoveder har for længst fundet ud af, at nuller foroven udelukkende angiver den værdi, som skal hæftes på tallet foran dem - og at de mindre enheders cifre sagtens kan gøre det samme. Vi kan altså erstatte nullerne med angivelserne for de mindre enheder:

5234

Hvis der nu ikke havde været nogle tiere i denne mængde, kunne vi have løst problemet på denne måde:

52 4

men så kunne der være tvivl om, hvorvidt det drejede sig om samme mængde som før, eller om 2 mindre mængder, som tilfældigvis er skrevet på samme linie. Vi har derfor besluttet os til at beholde de nuller, som angiver manglende repræsentation af en mængde, og derfor vil tallet blive - og bliver - skrevet på denne måde:

5204

4-tallet betyder altså, at vi har 4 x 1. 0 betyder 0 x tiere, som vi giver et sammensat symbol: 10. 2 betyder i dette tal, at vi har 2 x hundrede, hvilket er det samme som 2 x 10 x 10, mens endelig 5 betyder 5 x tusinde eller 5 x 10 x 10 x 10.

ANDRE TALSYSTEMER

Vi kunne meget let tænke os til et uhyre enkelt talsystem, som vi kunne kalde C-systemet.
Værdierne i dette C-tal system skulle repræsenteres ved symbolerne A og B, som ville betyde hhv. NUL og EEN.
Hvis dette talsystem skulle opbygges efter samme principielle metode, som vort netop beskrevne 10-tal system, ville den mindste enhed - større end ingenting - være EEN, og det ville vi i vort nye system skrive således:

B

Hvis der var een enhed mere end een, måtte vi til at benytte sammensatte symboler, og antallet TO ville vi skrive på denne måde:

BA

Ganske som i vort talsystem ville vi kalde denne værdi for noget ganske andet, end det, den faktisk er. Vi viser et 1-tal og et 0, hvorefter vi kalder det for TI.
I dette nye system viser vi et B og et A, og lige så logisk kalder vi det for C, hvor vi kan omregne til vort decimal-system, at C = 2 enheder.
Hvis vi ønskede at tælle i C-tal systemet, ville vi hurtigt komme op på 3 cifre, og vi kunne give denne mængde benævnelsen D, som naturligvis ville være det dobbelte af C.

2-TAL SYSTEMET

Datamater arbejder internt med svage spændinger, som i virkeligheden repræsenterer modsatte situationer som ja/nej - op/ned - alt/intet o.s.v.
Datamatens evne til at skelne mellem to klart adskilte værdier, gør det muligt at lave et talsystem, som er baseret på kun 2 forskellige cifre - ganske som det system, vi beskrev for lidt siden.
Dette talsystem kaldes for 2-tal systemet eller det binære talsystem. Ganske som i vort TI-tal system og i C-tal systemet har vi ingen 2-taller i det binære talsystem.
Da det binære talsystem blev introduceret til brug i datamater, forsøgte man at trække så meget som muligt fra vort nuværende 10-tal system med over - derfor blev værdierne i det binære system repræsenteret ved symboler, som vi allerede kender:

0

og

1

Anvendt alene har de samme værdi, som i 10-tal systemet, nemlig "ingenting" og "een".
Hvis vi skal vise, at mængden er på TO enheder, må vi benytte os af et sammensat symbol, og helt som i 10-talsystemet begynder vi forfra og skriver:

10

Da dette er et 2-tal system, vil nullet repræsentere enere, som vi her har ingen af, og 1-tallet repræsenterer antallet af enheder, som er 1 større end 1 - og den værdi kalder vi både binært og decimalt for 2.
Hvis vi ønsker at tælle binært, går vi frem efter samme principielle måde, som i 10-tal systemet:

0
1
10
11
100
101
110
111
1000
1001

Da det sidste tal repræsenterer en mængde på 9 enheder, ses det, at det binære talsystem anvender langt flere cifre end 10-tal systemet til at beskrive den samme mængde. Vi skal være glade for, at vi har 10-tal systemet til daglig, for det ville give nogle frygtelig store check-blanketter, hvis vi skulle skrive tallene i bogstaver også. Blot tallet NI, som i det binære talsystem skrives 1001, måtte staves således: *Een otter og een ener.* Hvis det drejede sig om værdien SYV-OG-TYVE, som binært skrives:

11011

måtte vi stave: *Een sekstener og een otter og een toer og een ener.* Med den inflation, vi har idag, skal vi være glade for 10-tal systemet.
Vi kan altså gøre en lang række sammenligninger mellem 10-tal systemet og

2-tal systemet, hvor det, som har størst betydning, er, at hver gang værdierne bliver 10 hhv. 2 gange større, sammenssættes tallet af endnu et ciffer.
Læst fra højre vil et tal i 10-tal systemet bestå af:

enere
tiere
hundreder
tusinder
o.s.v.

I det binære system har vi:

enere
toere
firere
ottere
sekstener
o.s.v.

I begge systemer vil et ciffer længere til venstre repræsentere kvadratet på dets nabo. Dette kan også udtrykkes ved potenser, idet 10 i 2. potens er 100, mens 2 i 2. potens er 4.

Det er derfor hyppigt, at man fremstiller talsystemerne som værende opbygget af grundtallet opløftet i 2. potens.
Om man ser det på den ene eller anden måde er som sådan underordnet, men vi er lidt kede af den foretrukne fremstilling, hvor man forsøger at beskrive det binære talsystem ved en decimal repræsentation.
Inden vi går i gang med de andre talsystemer, bringer vi en lille tabel over det binære og decimale talsystems repræsentationer for en række værdier:

Binært	Decimalt
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
10000	16
11111111	255

Byg selv: Z-80 datamat

Hobbydatamaterne befinder sig idag på samme stade, som Hi-Fi for 10 år siden: Udbuddet er begrænset, og hvis man ønsker kvalitet, er produktet kostbart. Ganske som for Hi-Fi dengang, må man bygge selv, hvis man ønsker at undgå ovenstående forhold. Et par unge mennesker besluttede sig til at gøre noget ved dette, og de har udviklet deres egen amatør-datamat.

Konstruktion og artikel:
Susanne Sønderstrup
Mogens Pelle

Der var engang -
Sådan begynder de fleste eventyr, men intet eventyr er så fantastisk, som den udvikling, vi har oplevet på hobbydatamatområdet.
I gamle dage - det vil sige for 3 år siden - var det den berømte MARK-8, alle talte om. Man købte printplader, der ganske vist var dobbeltsidede, men ikke gennempletterede, og det var et enormt besvær at få maskinerne til at virke. Interesserede meldte sig til diverse brugergrupper, læste alt, hvad der var at få om disse maskiner og følte sig i det hele taget som medlemmer af en stor, international (dog mest amerikansk) bevægelse.
Siden da er det, som om det hele har skiftet karakter. Processorer købes idag færdige eller som byggesæt. Datamattidsskrifterne er hovedsageligt fyldt med annoncer eller BASIC-programmer, og det hele har fået et stærkt kommercielt præg.
Denne udvikling er vi nogle stykker, som længe har været kede af, og vi bestemte os derfor til at forsøge at udvikle et system, der i alle retninger var så amatørvenligt, som muligt. Ordet *amatørvenligt* har her mere end een betydning.
Først og fremmest skal omkostningerne være venlige - hvilket naturligvis vil sige små. Men dette må ikke gå ud over effektiviteten og mulighederne. Somme tider står disse forhold i direkte modstrid med hinanden - og så må man vælge det bedste kompromis.
Vi må også kræve tilstrækkeligt med software, idet udviklingen af maskinprogrammer er en meget langsom proces. Her ligger sikkert årsagen til den stigende interesse for high level sprog som f.eks. BASIC.
Systemet skal være enkelt, så det kan både bygges og fejlrettes af personer med begrænset adgang til måleudstyr.
Endvidere skal systemet så vidt muligt opbygges af standardkomponenter, da disse giver færrest leveringsproblemer, og ikke mindst skal systemet kunne tilsluttes det størst mulige antal forskellige ydre enheder.
Dette er blot nogle få af de betragtninger, vi har gjort og har forsøgt tillempet i det system, som vi nærmere vil præsentere i den nærmeste fremtid. Denne artikel vil dække de nærmere begrundelser for de store linier i systemet, hvorimod de senere artikler bliver en beskrivelse af

og byggevejledning for systemets enkelte dele.

VALG AF CPU

Der findes efterhånden mange CPU'er på markedet - og hvert halve år kommer der en ny. De fleste af disse kan hurtigt lades ude af betragtning, alene på grund af den begrænsede mængde software, der er til rådighed. Tilbage bliver der faktisk kun 4, nemlig 6800, 6502, 8080 og Z-80.

6502 har vi slet ikke kunnet få nogen dokumentation om, da systemet skulle udvikles (dette er desværre stadig et hyppigt problem her i landet), og den måtte derfor lades ude af betragtning.

8080 er - selv med brug af hjælpekredse - relativt kompliceret rent hardwaremæssigt. Desuden er dens interne hukommelse dynamisk, hvilket vanskeliggør visse koblinger. Til gengæld findes der enorme mængder software, den er almidelig kendt, let at skaffe, og familien er godt udbygget med mange periferi-kredse.

6800 er let, rent hardware-mæssigt, men den har også dynamisk hukommelse. Der er et rimeligt udbud af software og også mange periferi-kredse.

Z-80 er let - rent hardware-mæssigt - og fuldkommen statisk. Ellers minder dens CPU mest om en sammenblanding af de 2 foregående. Den har 8080 instruktionssæt og (næsten) 6800's adresseringsformer. Hertil kommer et stort antal nye instruktioner, hvoraf mange er særdeles anvendelige. Den er dyrere end de 2 andre og ikke så kendt i detaljer, selvom de fleste har hørt om den. Udvalget af periferi-kredse er begrænset, men kredsene beregnet til 6800 og 8080 kan anvendes uden særlige problemer.

En nærmere sammenligning af disse 3 processorer har allerede været udført i adskillige artikler i udenlandske blade, og vi skal derfor ikke i denne omgang gentage successen.

OPBYGNING

Blandt radioamatører taler man ofte om *fuglereder*. Nogle af de billeder, man har set af datamat-opstillinger, kan vel også bedst karakteriseres på den måde.

Det er imidlertid et fænomen, der absolut bør undgås. Dels ser det ikke pænt ud, og dels er det vigtigt for datamatens funktion, at hele opbygningen er så stabil som muligt. Det med det pæne er et hurtigt løst problem. De færreste amatører har mulighed for og evner til at bygge et pænt kabinet. Derfor bør datamaten være beregnet til indbygning i et færdigkøbt standardkabinet.

Den mekaniske opbygning er et større problem, der strækker sig lige fra spørgsmålet om wire-wrap contra print, og frem til udvælgelsen af bestemte kredse til bestemte kort. Lad os på nuværende tidspunkt nøjes med at omtale de større linier - så kommer detaljerne, når vi tager fat på de enkelte kort.

SAMLINGSMÅDE

I praksis er der vel kun 2 måder at bygge en datamat op på: Wire-wrap og trykte kredsløb (PC). Wire-wrap er billigst, når man først har værktøjet, men uanset ens instrumentpark må man være indstillet på en større hovedpine i forbindelse med evt. fejlsøgning.

PC'er er dyrere og mere besværlige for konstruktøren. Til gengæld kræver samlingen af et ordenligt konstrueret print ikke andet end omhu og kendskab til lodning - så skulle man være sikret et godt resultat.

Der kan selvfølgelig opstilles mere avancerede betragtninger over disse 2 monteringsmetoder, men dem skal vi nok holde for os selv i denne forbindelse. Men vi blev enige om, at wire-wrap er en uheldig samlingsmåde for amatører - især på grund af fejlfindingsproblemet.

Den næste række af beslutninger var lettere at tage, og vi tror, at alle, der i de go'e gamle dage har forsøgt at samle en MARK-8, vil give os ret. Printplader til datamater bør være dobbeltsidede med gennempletterede huller, påtrykte komponentplaceringer og forgyldte kantkontakter i normal industriudførelse.

Vi er klar over, at dette betyder relativt høje priser, og vi har derfor undersøgt,

Lær Programmering

Programmering af datamater virker for mange som en mystisk kunstart, som er forbeholdt de få indviede. Der er dog intet mysterie knyttet til programmering overhovedet. Programmering er tværtimod det mest eksakte, som findes, og det er uhyre udviklende for ens personlighed at beherske dette fag. Man lærer at angribe problemer på den rigtige måde og løse dem på den letteste facon. Følg derfor med fra starten.

Programmering betyder planlægning, og det er nøjagtigt, hvad det er.
Enhver datamat må betegnes som direkte underbegavet, indtil et mere eller mindre fornuftsbetonet program får den til at reagere noget mere rimeligt.
Næsten alle kan lære at programmere, især hvis sigtet er mere avancerede sprog som BASIC, FORTRAN og hvad de nu allesammen hedder. Det er ikke alle, som kan blive lige hurtige til at fremstille programmer, og det er i virkeligheden ret få, som nogensinde bliver både effektive og erfarne i maskinkodning.
Effektiviteten betyder heldigvis ikke så meget til hobbybrug, hvor ethvert selvfremstillet program omfattes med langt større brugerglæde, end selv det mest avancerede program, som man har købt.
Disse sider vil sætte enhver, som kan læse indenad i stand til at lære at programmere.

I første omgang arbejder vi med en teoretisk datamat, som er meget nem at have med at gøre, idet dens instruktionssæt kun indeholder omkring 20 forskellige ordrer, og heraf vil vi i begyndelsen endda kun benytte os af de første 10.
Senere vil vi divergere ud fra denne teoretiske model til praktiske eksempler på både maskinkode og high level sprog.
De, der allerede har lært programmering fra "professionel" kilde, vil næppe kunne få meget ud af disse indledende sider, mens mange "selvlærte" sandsynligvis vil kunne finde et enkelt guldkorn her og der.

VOR TEORETISKE DATAMAT

Vi er så heldige, at vi til vor rådighed har et splinternyt eksemplar af den uhyre avancerede *Telemat*, som vi vil anvende til vor programmeringsundervisning.
Dette nyudviklede produkt har bl.a. den

egenskab, at det til enhver tid opfører sig nøjagtigt, som vi definerer. Hvilket er en enorm fordel.
Så lad os beskrive vor Telemat.
Telematen er forsynet med et endnu ikke defineret antal ind- og udgange til brug for ind- og udlæsning af data og ordrer. Vi har i første omgang tilsluttet en elektrisk skrivemaskine til vor Telemat, og den ind- og udgang, som skrivemaskinen er tilsluttet, benævner vi med tallet 01. Da der er to cifre (decimal) er der åbenbart mulighed for tilslutning af op til 99 forskellige ydre enheder, men det får vi næppe brug for.
Skrivemaskinen er tilsluttet Telematen på så praktisk en måde, at vi uden videre kan indskrive ordrer til dens lager.
På skrivemaskinen har vi endvidere to ekstra taster, som er meget vigtige:

START
STOP

Telematen er således indrettet, at når vi trykker på STOP, standser den øjeblikkelig med sit nuværende program og går op til lager ord nr. 000, hvor den afventer yderligere ordre.
Når maskinen er standset, kan den kun sættes i gang med START, som vil få Telematen til at udføre den ordre, som den er standset ved.

INSTRUKTIONSSÆTTET

Vi begynder med at gennemgå de første 10 af Telematens instruktionssæt.

LÆS, n Der indlæses fra indgang nr. n til akkumulatoren.

SKRIV, n Tilsvarende vil denne ordre flytte indholdet af akkumulatoren til udskrivningslinie nr. n.

GEM, n Indholdet af akkumulatoren gemmes i lagercelle nr. n. Evt. tidligere indhold i n ødelægges.

HENT, n Indholdet af lagercelle nr. n placeres i akkumulatoren. Evt. tidligere indhold ødelægges.

ADD, n Indholdet af lagercelle nr. n adderes til indholdet af akkumulatoren, og resultatet står i akkumulatoren.

SUB, n Der udføres en subtraktion af lagercelle n fra akkumulatoren. Resultatet står i akkumulatoren.

TÆL, n Indholdet af lagercelle nr. n forøges med 1.

HOP, n Programmet hopper til den ordre, som står i lagercelle n og begynder straks at udføre denne.

OMNUL Den ordre, som følger efter denne, udføres kun, hvis indholdet af akkumulatoren er lig med 0.

STOP Programmet standser. I modsætning til STOP-knappen på skrivemaskinen vil denne ordre få maskinen til at standse i selve STOP-ordren.

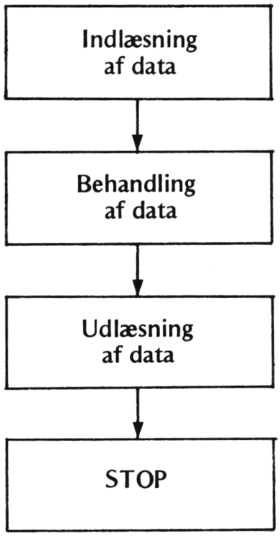
Der er yderligere en halv snes instruktioner til vor Telemat, men dem tager vi efterhånden, som vi får brug for dem.
Vi skylder nok også lige at forklare, at akkumulatoren er et register, som kan opfattes ganske som en lagercelle blot med den forskel, at alle operationer udføres i forbindelse med akkumulatoren.
Ovenstående instruktionssæt ser måske lidt primitivt ud, men bare vent - det er ikke småting, vi kan gøre med denne maskine.

DET FØRSTE PROGRAM

Som tidligere nævnt er vor Telemat en uhyre avanceret datamat, som er meget let at betjene. Vi har tændt for strømmen, og et tryk på vores STOP-knap på

skrivemaskinen vil få datamaten til at stille sig afventende i ordre nr. 000.
Når datamaten står i denne position, vil vi kunne indlæse data og ordrer til dens lagerceller, idet den først indtastede instruktion placeres i celle nr. 000, næste instruktion i celle nr. 001 o.s.v.
Dette benytter vi os af, idet vi ønsker at indlede med det simpleste af alle programmer: Sammenlægning af 2 tal.
Selvom denne opgave er meget simpel, vil vi alligevel lave et blok-diagram over funktionerne i programmet.
Blokdiagrammet er en stærkt simplificeret udgave af programmet, og ved blot lidt større programmer er det næsten en nødvendighed.

BLOKDIAGRAM



Dette blokdiagram kan i virkeligheden benyttes til praktisk talt alle programmer, men større programmer vil kræve mere detaljeret opløsning af de enkelte funktioner.
I dette tilfælde vil vi bygge vores enkle program op over ovenstående blokdiagram.

PROGRAM

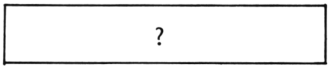
LÆS, 01	Første tal læses ind i akkumulatoren.
GEM, 006	Gem det første tal i lagercelle 006.
LÆS, 01	Et nyt tal indlæses til akkumulatoren.
ADD, 006	Indholdet af celle nr. 006 adderes til indholdet af akkumulatoren.
SKRIV, 01	Indholdet af akkumulatoren udskrives.
STOP	Datamaten stopper.

I vores Telemat vil enhver komplet ordre indtil videre kunne gemmes i en enkelt celle, så disse instruktioner vil blive lagret i celle 000 - 005 incl.
Hvis vi forestiller os cellerne placeret på række som linierne på en strimmel fra en regnemaskine, vil begyndelsen af lageret se således ud:

celle nr.	Indhold
000	LÆS, 01
001	GEM, 006
002	LÆS, 01
003	ADD, 06
004	SKRIV, 01
005	STOP
006	?
007	?

Bemærk, at der på nuværende tidspunkt er sat spørgsmålstegn ved celle nr. 006 og 007. Da vi netop har tændt maskinen og intet placeret i disse - eller efterfølgende celler - ved vi ikke, hvad der gemmer sig i disse.
Vi har også en akkumulator, og i dette øjeblik, hvor vi har afsluttet indtastning-

en af programmet, ved vi heller ikke, hvad den indeholder.
Vores akkumulator repræsenterer vi som en alm. celle, men med bogstavet A istedet for et celle-nr.



Datamaten er forsynet med en indre tæller, som sørger for, at en følgende instruktion bliver den næste, som udføres. Denne **Program Tæller** vil vi vise som en pil, der peger på den celle, som datamaten arbejder med i det pågældende øjeblik - altså den instruktion, som den er ifærd med at udføre.

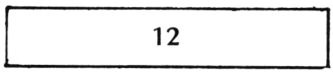
UDFØRELSE AF PROGRAMMET

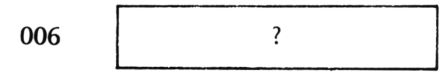
Vi er færdige med at indtaste vores program, og Program Tælleren står ved den sidst berørte celle (nr. 005) og afventer yderligere besked. Vi trykker derfor på STOP, hvilket får program tælleren til at pege på celle nr. 000. Næste skridt er et tryk på knappen START.
Det kan ikke undre nogen, at datamaten begynder med at udføre instruktionen, som står i celle nr. 000.
Udførelse af LÆS instruktionen indledes med en afventning, idet datamaten venter på vores tal, som den skal læse ind. Vi tager et tilfældigt tal og indtaster

12

Det kan ikke ses, men tallet "12" er fulgt af en vognretur, som fortæller datamaten, at hele tallet er indtastet - det kunne jo have været på flere cifre, så vi er nødt til at fortælle maskinen, at med vognreturen er tallet slut.
Lad os se på vores lagerceller efter udførelsen af den første instruktion.

Programtælleren står udfor celle nr. 001 og peger således på den næste celle, som har en instruktion, der skal udføres.
Det, der er interessant, er indholdet af akkumulatoren og celle nr. 006:

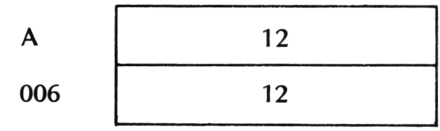




Vi ser, at vores LÆS ordre har virket, og det indtastede tal er nu gemt i akkumulatoren. Alt andet er uændret. Vi lader datamaten udføre næste instruktion:

GEM, 006

Vi ser igen på akkumulatoren og celle nr. 006:



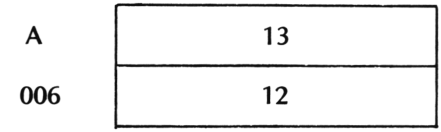
Indholdet af akkumulatoren er nu flyttet til celle nr. 006 - uden at indholdet af akkumulatoren er ændret. Dette skal vi erindre os, da det ofte kan give morsomme resultater, hvis en formodet tom akkumulator i virkeligheden indeholder information fra en tidligere ordre.
Vi lader Telematen udføre den næste instruktion:

LÆS, 01

Denne instruktion er identisk med den første, så vi ved, at der intet sker, før vi indtaster et tal, som sluttes med en vognretur:

13

Vi ser på indholdet af akkumulator og celle nr. 006 efter denne instruktion er udført:



Nu har vi begge de ønskede tal inde i datamaten, og vi er derfor klar til at lægge tallene sammen. Da vores ADD-ordre vil tage indholdet af en lagercelle og addere dette til indholdet af akkumulatoren og gemme resultatet samme sted, kan vi ordne dette med en enkelt instruktion:

ADD, 006

Og dette giver følgende indhold i registerne:

A	25
006	12

Nu bemærker vi igen, at indholdet af celle nr. 006 er uændret efter additionen, mens tallet 13 er tabt for evigt.
Vi vil derfor erindre os, at hvis vi skal bruge det oprindelige indhold af akkumulatoren igen efter en addition - eller anden operation - må vi gemme indholdet forinden i en lagercelle.
Nu skal vi have programmet til at udføre sin næste ordre:

SKRIV, 01

Indholdet af akkumulator og lagercelle 006 er stadig uændret efter denne ordre, men vores skrivemaskine har pludselig skrevet tallet 25 - hvad vi selvfølgelig også havde regnet med.
Den sidste ordre i dette program er en simpel STOP-ordre, som får datamaten til at standse. Hvis ikke denne STOP-ordre havde været placeret her, ville datamaten have fortsat med at udføre indholdet af celle nr. 006, som om det havde været en instruktion, og det kunne der være kommet mange pudsige ting ud af. Alle programmer må nøje definere en afslutning på programmet, så den slags undgås.

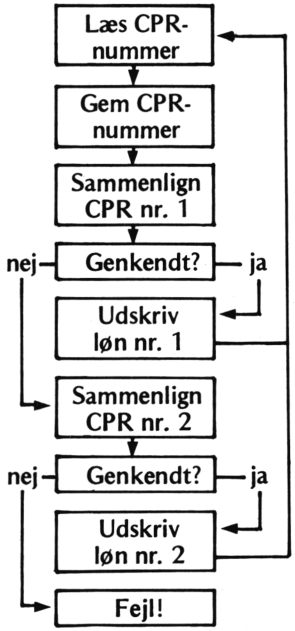
SKRIDT FOR SKRIDT

Det, som er vigtigt at notere sig lige fra starten, er, at datamaten skal have besked om samtlige detaljer i den samlede operation.
Hvis man programmerer i high level sprog, kan man ofte slippe lidt lettere, idet en enkelt ordre ofte udløser adskillige maskinkoder. Omvendt vil man i de rigtige maskinkoder (vores teoretiske model ligger sådan omtrent midt imellem) som regel være nødt til at definere endnu flere enkelte trin, idet hver maskinkode kun udløser meget simple operationer.
Hvis ovenstående program havde kørt på en rigtig datamat, havde man slet ikke bemærket, at der havde været brugt tid til de enkelte operationer, idet maskinen havde brugt hovedparten af tiden til at vente på tal.
I samme brøkdel af et sekund, som den sidste vognretur havde afsluttet tallet 13, havde skrivemaskinen udskrevet resultatet: 25. Selvom vi havde bedt maskinen addere indholdet af celle 006 i alt 1000 gange, ville dette være overstået, før skrivemaskinens valse var i ro efter vognreturen.
Det er denne hastighed, som er datamaternes store styrke, men samme hastighed kan også til tider give problemer, idet en programfejl kan betyde, at maskinen har cyklet adskillige gange rundt i lageret, før man bemærker, at der er noget galt.
Nogle fejl er ret uskyldige og resulterer blot i fejl i facit, mens andre fejl kan få som resultat, at programmet ødelægger sig selv, før årsagen opdages - og når programmet først er ødelagt, kan det være svært at finde fejlen.
Derfor gælder det i al programmering om at udvise den største omhu med en næsten sygelig sans for alle detaljer.

SPØRGSMÅL OG HOP

Det er datamatens evne til at udføre instruktioner, som indeholder spørgsmål og deraf betingede ordrer, der gør disse elektroniske maskiner så alsidige.
Vi vil derfor se, hvorledes vi kan løse en lille opgave ved hjælp af denne form for instruktioner.
Vi opfatter vor opgave som værende en del af et større program (et sådant "under"program kaldes for en Sub-rutine), og dette hovedprogram har lagret en del informationer til os.
Disse informationer er lønningerne til 2 medarbejdere, og medarbejderne er naturligvis kendetegnet ved deres CPR-numre.
Programmet skal afvente indtastning af

et CPR-nummer, hvorefter det tilsvarende løntilgodehavende skal udskrives på skrivemaskinen.
Da vi af erfaring ved, at operatører ofte taster forkert, skal programmet samtidig have indbygget en sikring mod fejl, idet en indtastning af et ikke genkendt CPR-nummer skal få datamaten til at give besked og stoppe op.
Sidstnævnte funktion tager vi med allerede nu, da det er vigtigt at imødegå alle tænkelige situationer ved at fortælle datamaten, hvordan, den skal handle i enhver situation.
Vi laver først et enkelt blokdiagram, som denne gang indeholder nogle flere funktioner:



Celle nr.	Instruktion/data
000	LÆS, 01
001	GEM, 016
002	HENT, 018
003	OMLIG, 016
004	HOP, 011
005	HENT, 020
006	OMLIG, 016
007	HOP, 013
008	HENT, 017
009	SKRIV, 01
010	STOP
011	HENT, 019
012	HOP, 014
013	HENT, 021
014	SKRIV, 01
015	HOP, 000
016	(arb.register)
017	(fejl)
018	(CPR nr. 1)
019	(løn nr. 1)
020	(CPR nr. 2)
021	(løn nr. 2)

Bemærk, at vi allerede nu har indført en ny instruktion, **OMLIG**.
Denne ordre har den betydning, at den følgende ordre kun udføres, hvis indholdet af akkumulatoren er identisk med indholdet af den lagercelle, hvis nr. følger efter OMLIG-ordren. Altså en instruktion, som minder meget om den allerede omtalte OMNUL-ordre.
Prøv selv at sammenligne programmet med blokdiagrammet, og løb programmet igennem, før du læser vores gennemgang. Cellerne markeret med parentes indeholder information, som svarer til beskrivelsen i parentesen.

Datamat amatører

Jan Lassen, tlf. (01) 71 63 99
Finsensvej 137, II
2000 Kbhvn F.
Endnu intet udstyr, men søger kontakt med andre, som er på begynderstadiet.

Erik Østergård
Amager Landevej 108
2770 Kastrup
Arbejder med en 8080 m. Floppy Disc og 36K lager, teletype og TV-terminal.

Alex Jessen, tlf. (03) 36 54 08
Dommervænget 101
4000 Riskilde
Indehaver af M6800 micro-datamat m. TV-terminal og kassette. Interesserer sig mest for software.

Gert P. Jørgensen, tlf. (09) 92 19 95
Hørmarken 5
8362 Hørning
Har lokalerne, men ingen datamat og hører derfor gerne fra andre i den modsatte situation.

Tom Hansen,
Dybbølgade 22, st.
6500 Vojens
Mangler alt undtagen interessen og vil derfor gerne i kontakt med evt. en klub om bygning af egen datamat.

Christian Rotenburg
Sønderbyvej, Daler
6280 Højer
Har intet udstyr, men interesserer sig for både hardware og dets praktiske anvendelse

John-Kjell Hoset, tlf. (02) 84 66 78
Haskollvn 30 D
N-3400 Lierbyen. NORGE
Har endnu intet udstyr, men er gerne med til at danne en norsk klub.

Björn Anfindsen, tlf. 29 45 41
Boks 2512
N-5001 Bergen, NORGE
Hører gerne fra norske nordmænd med interesse for datamater.

Svend Broholm, tlf. (02) 85 65 00
Mølleåparken 63
2800 Lyngby
Har både HP 27 og TI 59 og har derfor naturlig interesse i programmer til både HP og TI - både personligt og på evt. klub-basis.

E. Palsbo, tlf. (02) 65 51 30
Jordbærvangen 49
2760 Måløv
Indehaver af M6800 og deltager gerne i klubarbejde indefor både soft- og hardware.

Ole. T. Christensen, Tlf. (09) 12 67 51
Ingrids Alle 3
5250 Odense SV.
Intet udstyr, men interesse for både soft- og hardware. Gerne kontakt med selvbyggere omkring Silog Z 80.

Ole Malling, tlf. (01) 54 34 14
Burmeistergade 13
1429 Kbhvn K.
Indehaver af SC/MP - gerne kontakt.

Iver Nørgård, tlf. (01) 26 20 26
Gl. Kalkbrænderivej 68
2400 Kbhvn Ø.
Er interesseret i at bygge store, billige statiske lagre.

Poul Erik Andersen, tlf. (09) 80 14 19
Lindebjerg 21
5474 Veflinge
E & L Mini micro designer med interesse for både soft- og hardware ønsker kontakt med amatører og klubber.

P-T Aasestrand, tlf. (05) 16 08 09
Stor Hammeren 31
N-5033 Fyllingsdalen, NORGE
Er i besiddelse af en KIM-I og interesserer sig for både datamater og amatørradio. Kendskab til systemprogrammering.

Christian Hendrup, tlf. (09) 16 95 20
Elmelundsvej 4, v. 1408
5200 Odense V.
Har selv micro-datamat m. TV-skriver og arbejder helst i FORTRAN.

H.H.Bergstrøm Møller
Snerlevangen 4
2680 Solrød Strand
Ejer MEK 6800 D2 og er TP-Systemprogrammør, men mangler hardware erfaring. Vil gerne møde andre omkring et aftenskolehold m. undervisning i 6800 programmering plus hardware.

Så nåede vi bunden af stakken for denne gang, så vi modtager meget gerne flere kort fra datamat-amatører, som gerne vil i kontakt med ligestillede.
Af hensyn til bladets opbygning vil vi helst bringe 2 sider ad gangen, så vi kan risikere, at der ingen datamat-amatører er med i næste nummer.
Vi ved, at der allerede er optaget mange gode kontakter, og vi er interesserede i at fortælle om de erfaringer, som I gør med hinanden, både i klubarbejde og mere sporadisk.
Fat derfor pennen, og lad os tilflyde et par bevingede ord - så lægger vi papir og sværte til.

PROGRAM
Firma-likviditet

Man går ofte ud fra, at de mindre programmerbare lommeregnere i kraft af deres begrænsede antal instruktioner og lagerkapacitet kun kan benyttes til simplere formål som f.eks. beregning af matematiske formler og lign. Selvfølgelig har disse maskiner deres begrænsning, men efterfølgende program viser, hvor meget, der kan gøres med kun 49 instruktioner og 8 dataregistre. Det er samtidigt et praktisk program, som forretningsfolk kan have glæde af.

Likviditet angår os alle, og utallige forretningsfolk sidder ud til de små timer og beregner, og denne omsætningsændring kan betale sig, og om de faste omkostninger er for store, og om en ændring af kreditforholdende kan overleves. Beregnet for en enkelt uge er dette såmænd ikke så vanskeligt, men hvis alle forhold skal tages med i betragtning, og hvis tallene ændrer sig for hver periode, kan det blive en langsommelig opgave at beregne resultatet for forskellige muligheder.
Dette program, som er beregnet til en HP-25, gør disse beregninger uhyre enkle, og det tager ingen tid at opstille statistiske modeller for resultatet af de forskellige faktorers indvirken.

UDGANGSPUNKTET

Der er en række oplysninger, som er nødvendige for disse beregninger, og da de fleste forretninger køber ind til en længere periodes salg, regner vi med 2 forskellige perioder: En salgsperiode og en købsperiode.

Det står enhver frit for at fastlægge disse perioder som dage, uger, måneder eller år, men vi har i det praktiske eksempel regnet med en virksomhed, som køber ind for 4 uger ad gangen, og hvor salget forløber jævnt gennem disse 4 uger. Vi lader altså virksomheden sælge 1/4 af sit lager hver uge, og hver 4. uge indkøbes et nyt varelager.

Vi skal også have informationer om de forskellige betalingstidspunkter, da evt. kreditter har stor betydning for likviditeten. Vi arbejder i programmet kun med betalinger, og vi er således ligeglade med leveringstidspunkter, men leveringen er naturligvis afgørende for, om indbetalingerne sker samtidigt med udbetalingerne.
Den faktor, som vi benævner betalingstidspunktet, er således antal salgsperioder, som ligger mellem 1. betaling for købte varer og modtagelse af betaling for solgte varer. Hver gang programmet ser, at betalingstidspunktet har værdien 0, foregår der en udbetaling. Indbeta-

lingerne sker også, når en værdi er på 0 og derefter.
Hvis vi forestiller os, at der udelukkende handles kontant, og at salget begynder samme dag, som varerne hjemkommer, sættes begge de pågældende tidspunkter til 0. Det betyder, at betalingstidspunkt og uge nr. begge starter på 0.
Indbetalingerne for solgte varer sker for første gang i uge nr. 0, så hvis vi er nødt til at betale vore varer 2 uger, før vi får penge ind, sætter vi uge nr. til −2. På den måde går der 2 uger, før vi ser de første indbetalinger.
Omvendt kan det tænkes, at vi har kredit på købssiden - i så fald skal vi udskyde betalingstidspunktet.
Dette gøres ved at sætte betalingstidspunktet til mindre end 0. Hvis vi har 2 ugers kredit, sætter vi betalingstidspunktet til −2. Der går således 2 uger, hvor vi slipper for at betale.
I praksis betyder dette, at den ud- eller indbetaling, som først forekommer, skal være lig med 0, mens det andet betalingstidspunkt sættes negativt med lige så mange perioder, som forskellen i betalingstidspunkter er.
Der er også noget som hedder faste omkostninger. Disse fratrækkes den likvide beholdning hver uge uanset de øvrige betalingstidspunkter. Hvis man har kredit på sine faste omkostninger, må man gøre et indgreb i sin likvide beholdning fra starten. Det er desværre ikke muligt også at justere betalingstidspunktet for de faste omkostninger.
Nu nævnte vi den likvide beholdning, som det hele drejer sig om, og vi har mulighed for at indtaste en startkapital, som programmet så gør større eller mindre gennem perioderne.
Endelig er det muligt, at der sker omsætningsændringer, og disse styres ved et faktor-tal, så tallet 1 betyder uændret omsætning. Hvis omsætningen skal stige med 30 %, sættes faktor-tallet = 1,3.

LØBENDE ÆNDRINGER

Det er naturligvis muligt at ændre på alle disse faktorer, efterhånden som et tidsforløb gennemkøres. Der er dog små forskelle, hvormed programmet reagerer på de forskellige ændringer.
Hvis der laves en ændring i omsætningen, registreres dette først, når næste køb foretages, da det normalt ikke vil være muligt at øge omsætningen med den samme lagerbeholdning.
Det er også muligt at lave ændring i det antal perioder, som salget af eet indkøb strækker sig over, og dette sker blot gennem ændring af antal perioder.
De faste omkostninger kan selvfølgelig også ændres, og dette skal gøres udefra, da disse ikke justeres gennem ændring af omsætningen.
Endelig er det muligt at forøge egenkapitalen eller formindske den ved ganske simpelt at indskyde eller hæve de ønskede beløb på konto 7.
Men lad os se på selve programmet.

ordre nr.		Instruktion	Forklarende kommentarer til programmet
01	f	FIX 7	Sæt udlæsning til maks 7 decimaler.
02		RCL 2	Vi begynder med de faste omkostninger.
03		STO −7	De faste ugentlige omkostninger trækkes fra den likvide beholdning.
04		RCL 4	Derefter ser vi på uge nr.
05	g	X<0	Er ugetallet negativt?
06		GTO 11	Uge nr. er negativ, og vi springer salget over.
07		RCL 5	Hent salgsværdien for hele perioden.
08		RCL 3	Hent antal uger, som indgår i perioden.
09		÷	Periodens salgsværdi divideres med antal uger.
10		STO +7	Værdien af ugens salg lægges til den likvide beholdning.
11		RCL 0	Nu henter vi information om betalingstidspunktet.

12	g	X≠0	Er betalingstidspunktet = 0 (eller forskelligt fra 0)?
13		GTO 16	Betalingstidspunktet var forskelligt fra 0, og vi udskyder betalingen.
14		RCL 6	Betalingstidspunktet var = 0, så vi henter det aktuelle beløb.
15		STO –7	Udbetalingen fratrækkes vor likvide beholdning.
16		RCL 4	Endnu en gang ser vi på uge nr.
17	g	ABS	Vi skal bruge tallet til udlæsning, så vi sikrer os, at det er positivt.
18		EEX	Vi angiver, at vi nu taler om eksponenter.
19		7	Og eksponenten er 7, altså = 1.000.000.
20		÷	Dette tal deler vi op i uge nr, som derved bliver 0,000000n.
21		RCL 7	Vi henter oplysningen om vor nuværende likvide beholdning.
22	f	INT	Og sikrer os, at der ingen unødvendige decimaler er med.
23	g	X≥0	Er den likvide beholdning lig med eller større end 0?
24		GTO 27	Det var den, og vi kan springe korrektionen over.
25		X↔Y	Den likvide beholdning var negativ, så vi trækker uge nr. ned.
26		CHS	Uge nr. gøres også negativ, så efterfølgende ikke går galt.
27		+	Nu kan vi addere den likvide beholdning og uge nr.
28		R/S	Vi standser op, så beholdning og uge nr. kan aflæses.
29		1	Vi skal bruge et 1-tal.
30		STO +4	Som vi tæller uge nr. op med.
31		RCL 4	Derefter henter vi det nye uge nr.
32		RCL 3	Og vi henter antal uger, som salgsperioden strækker sig over.
33		÷	Vi deler antal uger i salgsperioden med nuværende uge nr.
34	g	FRAC	Vi er kun interesserede i decimalerne (går divisionen op?).
35	g	X≠0	Hvis X er forskellig fra 0, gik divisionen ikke op.
36		GTO 39	Da divisionen ikke gik op, er vi ikke ved begyndelsen til en ny salgsperiode.
37		RCL 1	Vi henter faktor-tallet for omsætningsændringer.
38		STO x5	Salgsværdien ændres i henhold til omsætningsændringen.
39		1	Endnu en gang skal vi bruge et 1-tal.
40		STO +0	Som vi tæller betalingstidspunktet op med.
41		RCL 0	Vi henter det nye betalingstidspunkt.
42		RCL 3	Vi henter antal uger i salgsperioden.
43	f	X≠Y	Er betalingstidspunket forskelligt fra antal uger?
44		GTO 02	I så fald begynder vi blot forfra.
45		RCL 1	Da betalingstidspunktet passer med antal uger, henter vi oplysningen om evt. omsætningsændring.
46		STO x6	Købsbeløbet korrigeres i henhold til omsætningsændringen.
47		CLX	Nu skal vi bruge et 0.
48		STO 0	Vi sætter betalingstidspunktet = 0 (der var gået en periode).
49		GTO 02	Alt er nu klart til en ny uge, så vi begynder forfra.

Når programmet er vel indtastet i maskinen, skal de faste størrelser gemmes i de otte lagerregistre.

Betalingstidspunktet	**STO 0**	**Start i uge nr.**	**STO 4**
Omsætningsændring	**STO 1**	**Samlet salgspris**	**STO 5**
Faste omkost. pr. uge	**STO 2**	**Samlet købspris**	**STO 6**
Antal perioder pr. køb	**STO 3**	**Likvid beholdning**	**STO 7**

Hvis vi tager som udgangspunkt en virksomhed, som udelukkende handler kontant, har uændret omsætning, har faste omkostninger hver uge på kr. 100,00, køber ind til 4 uger ad gangen, betaler kr. 1.000,00 for hvert indkøb, sælger de samme varer for kr. 1.500,00 og har en likvid beholdning på kr. 1000,00, vil dataindtastningen se således ud:

```
    0  STO 0
    1  STO 1
  100  STO 2
    4  STO 3
    0  STO 4
 1500  STO 5
 1000  STO 6
 1000  STO 7
```

KØRSEL AF PROGRAMMET

Når både program om data er placeret i maskinen, trykkes på R/S og programmet går i gang.
Der går knap et sekund, hvorefter lommeregneren udlæser:

275.0000000

De sidste cifre omtaler uge nr., så vi er i uge nr. 0. Tallet før decimalpunktet er vores likvide beholdning efter udløbet af den første uge. Og det kan vi let kontrollere:

Likvidbeholdning	1000
– faste omkostninger	-100
Rest	900
Salg (1500/4)	375
Rest	1275
Varekøb	–1000
Endelig beholdning	275

Nu kan vi på forhånd meget let regne ud, at vi næste uge har en udgift på 100 og atter en indtægt på 375, hvilket skulle give en ny beholdning på 550. Vi trykker på R/S og denne gang går der et sekund, før lommeregneren svarer:

550.0000001

Hvilket selvfølgelig betyder, at vi efter uge nr. 1 har en beholdning på 550.
Vi lader programmet løbe et par uger og ser, hvad der sker:

825.0000002
1100.0000000

Her faldt uge nr. ud, da beholdningen er så stor, at der ikke er plads til udlæsningen. Hvis man bliver i tvivl om den faktiske uge nr. kan man fremkalde register nr. 4, som viser, at det drejer om uge nr. 3. Saldoen er selvfølgelig på 1100.
Vi tager en tur til:

375.0000004

Nu blev beløbet mindre (næste periodes køb blev betalt), og uge nr. er atter på sin plads yderst til højre.
Nu beslutter vi os til at gøre livet surt for firmaet, og vi sætter de faste omkostninger op til kr. 300 pr. uge:

```
  300  STO 2
```

Og vi tager et par uger til:

```
 450.0000005
 525.0000006
 600.0000007
– 325.0000008
```

Nu må vort firma lukke eller låne lidt penge i banken. Vi forøger beholdningen med yderligere kr. 1000, og vi sætter omsætningen i vejret, indtil videre med 10 % for hver uge:

```
1000  STO +7
 1.1  STO 1
```

Og så giver vi firmaet 4 uger mere at leve i. (Firmaet reducerede sine faste omkostninger med 100, men renter og afdrag på banklånet er andre 100, så vi har ikke reduceret de faste omkostninger.)

750.0000009
825.0000010

900.0000011
−87.0000012

Således slutter uge 12, og da −87 er mere end resultatet fra før, lader det altså til at gå bedre nu. Som et sidste eksempel vil vi nu forringe hans likviditet, idet han - tvunget af konkurrenterne - ser sig nødsaget til at give sine kunder 4 ugers kredit.
Da vi er nødt til at starte programmet forfra, og ugetælleren ikke virker i første halvdel af gennemløbet, sætter vi uge nr. til −3:

```
-3  STO 4
 0  STO 0
 f  CLR PRGM
```

Vi løber hurtigt gennem de næste par uger, og det ses direkte, hvor meget fremmed kapital, som er nødvendigt for at firmaet klarer denne ekstra kredit.

− 1487.000000 (uge −3)
− 1787.000000 (uge −2)
− 2087.000000 (uge −1)
− 1933.000000 (uge 0)
− 2990.000000 (uge 1)

Nu bliver direktøren desperat. Han skærer ned på reklamebudgettet, låner kr. 3000 i banken, og stopper omsætningsstigningen.

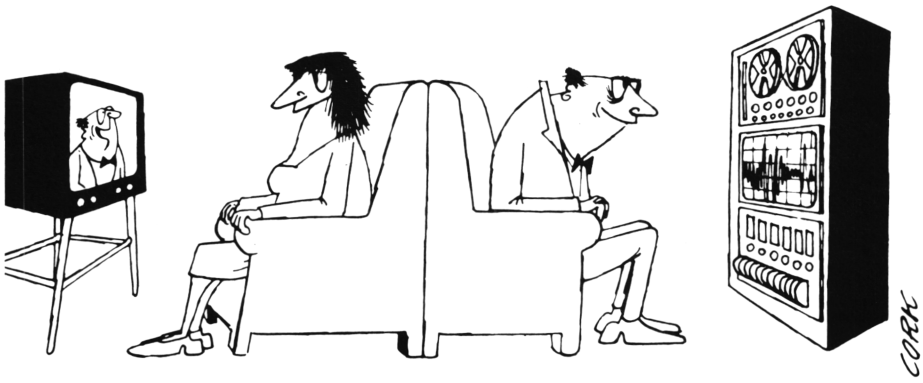
```
  1  STO 1
100  STO 2
3000 STO +7
```

Og det gav de resultater, som han længe havde drømt om:

363.0000002
717.0000003
1071.000000
215.0000005
568.0000006
922.0000007
1276.000001
420.0000009

Støt og roligt vil firmaet forhåbenligt komme på ret køl - takket være en lommeregner.
Nå, spøg til side. Programmet indeholder faktisk mange glimrende detaljer, som vi vil henvise til i forbindelse med vort programmeringskursus - derfor den lidt længere forklaring.

Til slut blot en praktisk detalje. Det kan virke lidt irriterende, at uge nr. til tider falder udenfor formatet. Dette kan rettes, hvis ordren i celle nr. 19 rettes til 6 (istedet for 7). Dette vil dog betyde, at man med mindre kassebeholdninger og små uge nr. risikerer et 0 efter uge nr.
Træerne får desværre ikke lov til at vokse ind i himlen. PH



Datamat kapacitet

Når der skal vælges datamat, gås der ofte ud fra mærkelige forestillinger om datamat kapacitet. Det er selvfølgelig ikke uvæsentligt at undersøge specifikationerne, inden man køber, men alt for ofte tillægges de forkerte specifikationer hovedvægten, mens de faktorer, som i virkeligheden er afgørende for ens fremtidige samvær med en datamat, lades i baggrunden. Det forsøger vi her at rette lidt op på. Vi begynder med de grundliggende krav.

Der er en række helt grundliggende krav, som nødvendigvis må stilles til enhver datamat uanset dens størrelse og hastighed:

Der skal kunne indlæses data i et nærmere specificeret format.

Der skal kunne foretages en behandling af disse data i henhold til programmerede instrukser.

Der skal kunne udlæses data i et nærmere specificeret format.

Det skal udefra være muligt at stoppe udførelsen af et program og bringe datamaten i en nærmere defineret situation.

Vi må altså - meget naturligt - kræve, at vi er i stand til at fodre vor datamat med informationer i form af data og instruktioner, at den behandler disse data i overensstemmelse med instruktionerne, og at den kan aflevere resultatet bagefter.

At vi også skal kunne stoppe maskinen under udførelse af et program er en absolut nødvendighed, da vi ofte vil kunne risikere, at datamaten på grund af fejl i programmerne begynder på mærkelige løkker, som den måske aldrig kan komme ud af igen. Normalt vil vi fortælle i slutningen af hvert program, at datamaten er færdig, og at den skal stille sig i en afventende position. Men vi må altså også kræve, at dette kan gøres udefra.

INDLÆSNING AF DATA

Det absolut væsentlige ved enhver funktion omkring datamater er absolut og endelig definition af alt, hvad der sker. Samtlige informationer, som datamaten beskæftiger sig med, er i binær form, d.v.s. en kombination af "0" og "1". De binære tal er selvfølgelig ikke rent fysisk gemt som små nuller og ettaller, som flyder rundt i maskinen, men som ændringer i spændinger, så ingen eller en lav spænding opfattes som "0", og en kraftigere spænding opfattes som "1".

Dette er den eneste form for data, som vor datamat er i stand til at behandle, og de informationer, som vi ønsker at give den udefra, må på den ene eller anden måde blive anbragt i datamaten i binær form.
Den absolut enkleste form for indlæsning af data foregår via omskiftere, som i kraft af deres position indikerer et "0" eller et "1".
Frontpanelet på en IMSAI 8080 er et glimrende eksempel på en sådan direkte indlæsning (indtastning) af binære data. Det er teoretisk muligt at indlæse enhver form for data og program via omskifterne på frontpanelet på en IMSAI, og kun ens tålmodighed sætter den effektive begrænsning.
Heldigvis findes der mere avancerede metoder til indlæsning af data, så disse både i format og hastighed kan anbringes i datamaten på mere hensigtsmæssig facon.

BEHANDLING AF DATA

De informationer, vi har anbragt i datamaten, vil normalt være en kombination af instruktioner og data, som skal behandles af maskinen.
For at den fornødne behandling kan foretages, må datamaten være forsynet med et sæt instruktioner, som hver resulterer i en nøje defineret operation.
Mange års erfaring med større datamater har lært os, hvilke instruktioner, som er de mest nødvendige, og hvilke operationer, det er mest hensigtsmæssigt at foretage gennem en programmeret kombination af enkelte instruktioner.
Maskinens instruktionssæt er fast indkodet i CPU'en (Den Centrale Styreenhed), og der kan ikke ændres ved disse instruktioner.
Instruktionssættet er kendetegnet ved den brug, vi kræver af datamaten, og der vil derfor altid være visse primære funktioner, som alle datamater er i

stand til at udføre, selvom instruktionerne kan have varierende benævnelser.
Instruktionerne opdeles i grupper efter deres funktion, og en hyppig opdeling kan se således ud:

Data ind- og udlæsning
Data transport
Data behandling
Programstyring

Det ses, at de 3 første grpper af instruktioner er helt analoge med de 3 første krav til vor datamat. Data transport står for de flytninger af data, som er nødvendige, for at maskinen kan behandle disse i sine registre.
Data-behandling inkluderer alle matematiske funktioner, op- og nedtællinger og sammenligninger.
Programstyringen omfatter alle de instruktioner, som anvendes til at kontrollere udførelsen af et program og inkluderer hop-ordrer og betingede hop-ordrer.
Det er strengt taget ikke nødvendigt med mere end en lille snes forskellige instruktioner, før man har en datamat, som opfylder vore grundliggende krav, hvilket vi bl.a. udnytter i forbindelse med vor programmeringsmodel.
Af hensyn til datamatens hastighed og bekvemmeligheden i programmeringen ser man dog gerne, at maskinen har et langt større sæt instruktioner til operatørens rådighed, men det er ikke særlig relevant udelukkende at bedømme datamaters kvalitet og kapacitet på baggrund i en simpel sammentælling af deres instruktioner.

UDLÆSNING AF DATA

Når datamaten har færdigbehandlet sine data, er det naturligvis vigtigt, at vi kan få resultaterne oplyst.
Maskinen skal derfor være i stand til at udlæse data på en sådan måde, at vi kan forstå de udlæste oplysninger.
Den enkleste form for udlæsning er brug af lysdioder, hvor hver diode repræsenterer et binært ciffer, og hvor en slukket diode er = "0", mens en tændt diode ret logisk står for et "1".
Dette er en form for udlæsning, som set fra maskinens side er overmåde praktisk, da datamaten som bekendt arbejder med binære tal.
Her er igen IMSAI 8080 et glimrende eksempel på en sådan form for data-udlæsning, idet hele dens frontplade er fyldt med lysdioder, som indikerer indholdet af registre og data-celler.
Naturligvis er en sådan form for udlæsning af data ikke praktisk, hvis det drejer sig om større mængder, og vi er også på dette punkt så heldige, at vi har langt mere avancerede metoder til vor rådighed.

PROGRAMSTANDSNING

Som tidligere berørt vil man ofte komme i den situation, at et nyt program får maskinen til at opføre sig højst ukontrolleret. Det er derfor absolut nødvendigt med en ydre stop-funktion, som bringer maskinen under kontrol af operatøren.
Denne funktion benævnes **Interupt**, og findes på alle maskiner i mere eller mere avanceret udgave.
Nogle datamater har temmelig komplicerede systemer opbygget omkring interupt funktionerne, så man bl.a. garanteres visse instruktioner eller sæt af disse udført, før maskinen standser og afventer yderligere instrukser.
Der kan bl.a. opstå problemer, hvis maskinen er ved at indlæse data, når den udsættes for et interupt. Flere datamater er derfor forsynet med programmerbare interupt funktioner, hvormed det er muligt at ændre det omfang af interupt, som datamaten vil acceptere under udførelse af programmet.
Hele problematikken omkring interupt er især kompliceret i forbindelse med store anlæg, hvor mange brugere på samme tid skal have adgang til datamaten. Den kan selvfølgelig ikke udføre mere end een ting ad gangen, og man må derfor prioritere de forskellige opgaver, så datamaten altid beskæftiger sig med det, som til enhver tid er mest påtrængende.
Til de mindre datamater, som vi for det meste beskæftiger os med, er det fuldt

tilstrækkeligt med en uhyre simpel interupt, som blot standser maskinen efter udførelse af den nuværende instruktion, hvorefter programtælleren sættes til den værdi, som operatøren indlæser. På den måde vil man til enhver tid være i stand til at styre datamaten manuelt.

DATAMAT KAPACITET

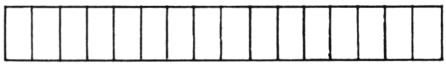
Det er kombinationen af en lang række faktorer, som bestemmer datamatens kapacitet.
Ordet "kapacitet" benyttes ligeledes i mange forskellige betydninger, og da de alle virker lige rigtige, skal vi ikke forsøge en mere differentieret definition af dette ord.
Vi skal istedet se på de faktorer, som hver for sig er bestemmende for den samlede kapacitet.
Disse faktorer inkluderer:

Ordlængde
Arbejdslagerets størrelse
Antal og arten af instruktioner
Antal og arten af tilslutningsmuligheder.
Datamatens hastighed
Programmernes kvalitet
Operatørens effektivitet

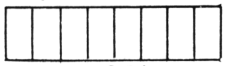
Normalt er det de fire første faktorer, der opfattes som værende de afgørende, men i virkeligheden er det med de relative små forskelle, som findes i dagens micro-datamater, de to sidste forhold, der spiller den altafgørende rolle.

ORDLÆNGDEN

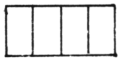
Hver gang CPU'en udfører en instruktion, bearbejder den flere binære cifre ad gangen. Hvert af disse cifre benævnes en **BIT** og disse bit samles i ord - eller som det hedder på engelsk: **Byte**.
Vi besluttede os for et par sider siden til at benævne datamaterne efter deres ordlængde, altså antal BIT pr. ord, og vi besluttede lige så bestemt, at en micro-datamat har en ordlængde på 8 bit - eller mindre.
Ordlængden er relativ vigtig, idet den er afgørende for, hvor store tal, datamaten



1 Byte å 16 Bit = max. 65.536 decimalt



1 Byte å 8 Bit = max. 256 decimalt



1 Byte å 4 Bit = max. 16 decimalt

Selvom micro-datamater har 4 - 8 Bit i hvert ord (Byte), arbejdes der i praksis næsten altid med 32 Bit i matematiske opgaver. Af disse 32 BIT benyttes 1 Bit til fortegnsangivelse, og der benyttes ofte 3 eller 4 Bit til angivelse af decimalpunktets placering. I micro-datamater vil man derfor som regel bruge 4 ord til hver konstant, som indgår i matematiske beregninger, mens en mini-datamat med ordlængde på 16 Bit kan nøjes med 2 Byte til hver konstant.

kan behandle i en enkelt operation. Da vi især beskæftiger os med 8-bit maskiner, kan vore datamater differentiere mellem 256 forskellige størrelser. D.v.s. at det største decimaltal, som vi kan behandle i binær form, er 256.
Hvis datamaten kan behandle 16 bit ad gangen, bliver tallet meget større, nemlig 65.536. De tilsvarende decimaltal for 24- og 32 bit datamater er henholdsvis 16.777.215 og 4.294.967.304.
Det ses heraf, at hvis vi ønsker at behandle større tal med en 8-bit datamat, må vi gemme tallet i flere ord, og at den deraf følgende mere komplicerede matematik nødvendigvis må tage længere tid.
Der er i praksis intet i vejen for selv meget store tal i en datamat med ord på 8 bit, men hver regneoperation kommer til at tage længere tid, da den skal udføres som en kæde af operationer.
Det er dog ikke desto mindre således, at da de fleste maskiner er så hurtige, at de kan udføre mindst 10.000 additioner i sekundet, vil man sjældent bemærke, om der bruges nogle få brøkdele af et sekund ekstra her og der.
Selvom ordlængden har betydning for hastigheden, vil den kun i programmer af statistisk art og med store tal have betydning af begrænsende art.

ARBEJDSLAGERETS STØRRELSE

En datamat benævnes ofte efter dens lagers størrelse. Normalt omtaler man antallet af ord (Byte), og man benævner maskinens lager efter antal ord gange 1.024. (2 i 10. potens = 1.024).
Således vil et lager på 1K være på 1.024 byte, mens et 64K lager er på 65.536 ord. I sjældne tilfælde tæller fabrikanterne ikke ord, men bit, og det kan jo hurtigt blive til noget.
Lagerets størrelse er ikke direkte afgørende for maskinens hastighed, men hvis lageret er for lille, er det nødvendigt at forøge det med f.eks. magnetbånd, og det er en langsommelig affære at ud- og indlæse data fra magnetbånd, når man sammenligner med hastighederne i maskinens interne lager.
De fleste micro-datamater kan idag udbygges til et 64K lager, så her adskiller de sig ikke væsentligt fra hinanden. Der kan være forskelle på de forskellige lagres hastighed og strømforbrug, men de lagre, som en fabrikant anbefaler til sin datamat, er normalt så hurtige, at det ikke er dem, der sætter grænsen for maskinens hastighed.

INSTRUKTIONSSÆTTET

De største forskelle på microdatamaterne finder man i antallet og arten af instruktioner, og det er da også som regel instruktionssættet, der af fabrikanten fremhæves som værende årsagen til især deres produkts fortræffeligheder.
Før man falder for dette, må man gøre op med sig selv, om man vil programmere i maskinkode eller High Level sprog som f.eks. BASIC.
Hvis man insisterer på at kode i maskinkode, må man være klar over, at man får noget at se til. Det er en langsommelig og vanskelig opgave at programmere blot mindre programmer i maskinkode, og chancerne for fejl i programmet er direkte proportionalt med antallet af instruktioner. Og i et maskinkodet program kan det let blive til nogle tusinde stykker, før man er færdig.
Erfaringsmæssigt bør man afholde sig fra at basere sit samvær med datamater på maskinkode, hvis man altså ikke lige akkurat finder denne form for hjernearbejde underholdende. Den smule tid, man kan vinde i hurtigere kørsel af programmerne, taber man tusinde gange i ekstra tid til programmeringen.
Hvis man insisterer på at arbejde i maskinkode, er detaljerne omkring instruktionssættet vigtige, og det gælder om at vælge sin CPU med største omhu. Vi vil i afsnittet CPU-Arkitektur opstille markedets forskellige CPU'er i skemaform, så de faktiske forskelle bliver lettere at se.
For de, der ønsker at programmere i f. eks. BASIC, er det mindre vigtigt, om CPU'en er i besiddelse af flere eller færre smarte detaljer, hvis ikke disse er udnyttet i den oversætter, som i datamaten laver BASIC-programmet om til den maskinkode, som datamaten kan forstå.

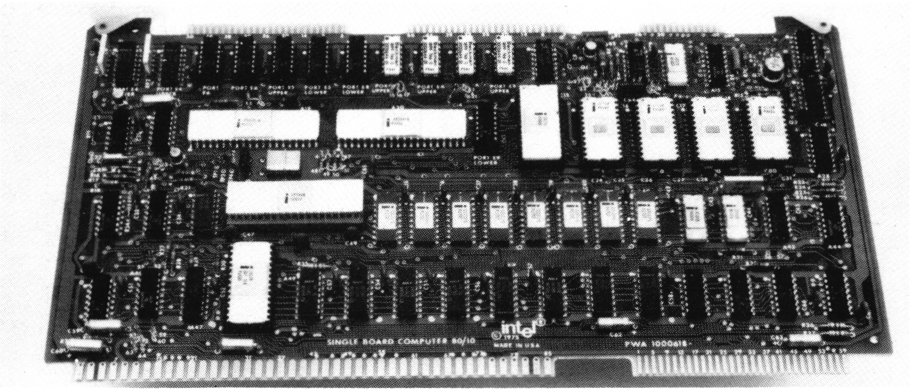
TILSLUTNINGSMULIGHEDER

Praktisk talt alle micro-datamater kan tilsluttes op til 256 forskellige ydre enheder på både ind- og udgangssiden.
Problemet omkring ydre enheder er således ikke af elektrisk art, men udelukkende mekanisk (er der plads til stik?) og programmæssigt betinget.
Der kan være mindre forskelle i de hastigheder, hvormed data kan overføres til f.eks. båndoptager, men i praksis er det de ydre enheder, som sætter disse begrænsninger.
Enkelte datamater med veldimensioneret strømforsyning kan også levere strøm til de ydre enheder, mens andre kræver ekstra strømforsyninger. Men dette kan højest blive et prisspørgsmål.
Yderligere forskelle, som især er af prismæssig art inkluderer udformningen omkring in- og output faciliteterne. Der findes datamater, som på et enkelt print indeholder elektronik for tilslutning af op til 4 ydre enheder, hvilket selvfølgelig giver et relativt dyrt print, mens det på den anden side kan blive dyrere at opnå samme totale antal tilslutningsmuligheder via flere, men mindre print, som kun kan acceptere 1 signalkilde.
I virkeligheden er der blandt dagens datamater ingen forskelle på mulighederne

for tilslutning af ydre enheder, som klart trækker det ene produkt frem for det andet.
Med hensyn til kapacitet - både hastighed og mængde - er disse forskelle i alle tilfælde ikke til at få øje på.

DATAMATENS HASTIGHED

Her er der forskelle, som er til at tage at føle på - og som kun vil blive bemærket af de færreste.
Når man omtaler datamaters hastighed, er det først og fremmest dens **Clock Impuls Frekvens**, som der tænkes på.
(Bliver grundigt behandlet i afsnittet om CPU-arkitektur)
Jo højere frekvens, jø større hastighed.
Typiske frekvenser for micro-datamater ligger mellem 1 og 3 MHz (**millioner Hz**) og enkelte datamater har mulighed for variation af denne frekvens.
Når disse forskelle i hastighed er til at tage og føle på, skyldes det naturligvis, at man ikke kan komme uden om, at en maskine med Clock frekvens på 2 MHz er dobbelt så hurtig som en maskine, der "kun" ligger på 1 MHz.
MEN - til hovedparten af de programmer, som datamat-amatører kommer til at arbejde med, sker der så megen kommunikation med ydre enheder, som er håbløse sinker i sammenligning, at enhver forskel i hastighed totalt udlignes.
Ofte er de langsommere maskiner endda hurtigere, idet de langsomme datamater er ældre (måske et helt år ældre), og det vil ofte have den virkning, at oversættere og andre nødvendige programmer er så meget bedre gennemarbejdet, at de udnytter maskinen langt mere effektivt.
Kun hvis datamaten skal benyttes til mange og lange beregninger med både kompliceret matematik og store talmængder, hvor kørselstiden bliver målt i timer, er der virkelig begrundelse for at vælge den hurtigst mulige CPU.
Men husk på - den nyeste teknologi er som regel også den dyreste. Det kan være en kostbar oplevelse at ville have det absolut sidste nye i denne branche.



Som det fremgår af de foranstående betragtninger, er datamat kapacitet et ret flydende begreb, som kan gøre det vanskeligt at vælge rigtigt.
Dette foto, f.eks., af Intel's SBC-80/10 er et glimrende eksempel på den lethed, hvormed begrebsforvirringen opstår. I dette tilfælde står SBC for Single Board Computer - hvilket kan oversættes til datamat på et enkelt print.
SBC-80/10 er opbygget over Intel's egen 8080A CPU, som også anvendes i IMSAI 8080, og 80/10'eren indeholder både hukommelse, CPU (nederst til venstre) og input/output kredse.
Men efter vor definition er dette altså en avanceret micro-processor, idet det ikke er muligt direkte at kommunikere med systemet. Dette gør afgjort ikke bedriften mindre

KLUBINFORMATION
Hvis du vil i kontakt med andre datainteresserede, skal du blot udfylde dette kort og sende det til os - vi bringer navnene i næste nummer af BIT.

FORSLAG TIL NYT NAVN
Skriv venligst dit forslag ud for "klubbens navn", og din afsender nederst.

Klubber åbne for medlemmer / interesseret i kontakt med andre klubber:

Klubbens navn: _____

Adresse: _____ Evt. tlf.: _____

Postnr.: _____ By: _____

Indmeldelsesgebyr, kr.: _____

Kontingent pr. måned, kr.: _____

Klubbens udstyr: _____

Speciel interesse: _____

Nuværende antal medlemmer: _____

Datamat-amatører, som er interesset i at blive kontaktet af klubber og andre datamat-amatører:

Navn: _____

Adresse: _____ Evt. tlf.: _____

Postnr.: _____ By: _____

Nuværende udstyr: _____

Speciel interesse: _____

Ønsker helst kontakt med: _____

KLIP LANGS DE FULDT OPTRUKNE STREGER
SENDES SOM BREVKORT, HUSK PORTO 80 øre

Undertegnede bestiller herved et abonnement på BIT.

Jeg ønsker oktober/november/december ... for kr. 25,00
Jeg ønsker 1 år (11 numre) ..... for kr. 75,00

Navn: _____

Gadeadresse: _____

Postnr.: _____ By: _____

Evt. udland: _____

Ovenstående bedes udfyldt i forbindelse med abonnementstegning.

Nedenstående oplysninger modtager vi meget gerne, da de vil hjælpe os til bedre at kende vore læsere. – Oplysningerne vil naturligvis ikke blive videregivet.

Alder: _____ Gift/ugift: _____

Stilling: _____

Jeg læser regelæssigt disse fagblade: _____

Jeg interesserer mig især for (f. eks. hardware, software etc.): _____

Jeg har et anlæg bestående af (f. eks. microdatamat, floppy-disc etc.): _____

Jeg påtænker at anskaffe (f. eks. microprocessor, linieskriver etc.): _____

Jeg vil især se frem til artikler omhandlende (angiv emner): _____

BREVKORT

Husk afsender

Porto
100
øre

Til:

Telepress ApS

Greve Strandvej 42
2670 Greve Strand

KLIP LANGS STREGERNE HELT TIL BLADETS KANT

BREVKORT

Husk afsender

Porto
100
øre

Til:

Telepress ApS

Greve Strandvej 42
2670 Greve Strand