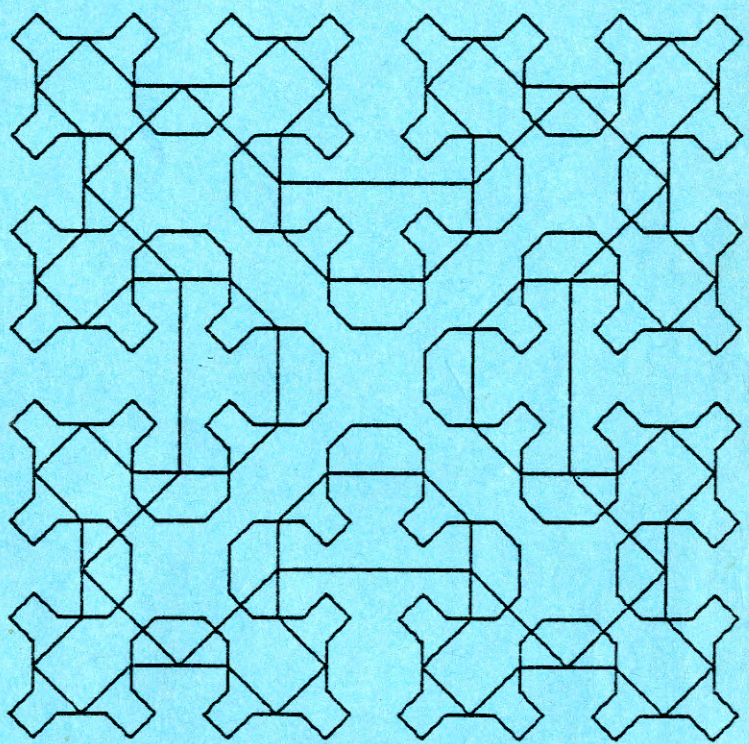
Ole Kamuark

DATA + ALGORITMER

DATALÆRE VED DE TEKNISKE SKOLER



Sierpinski kurver

Januar 1983 Nr. 1.

Indledning:

Hermed foreligger det første nummer af datalærebladet. I kraft af fagets udvikling og voksende betydning i de tekniske uddannelser, er vi en gruppe lærere, der finder det væsentlig, at der bliver etableret et forum, hvor den faglige og pædagoriske diskussion kan foregå og dermed styrkes. Vi har ligeledes en ide om, at et sådant blad kunne medvirke ved distribution af programmel og hardware komponenter, for således at fremlægge et bredt og varieret materiale til brug i undervisningen. Det er vores håb, at mange undervisere vil bidrage med beskrivelser af undervisningsforløb, for på denne måde at videregive erfaringer med anvendt indhold og metoder.
Med disse intentioner bliver det klart, at første udgivelse ikke kan leve op til disse krav. Redaktionen mener og arbejder på, at dette skal forbedres.
Vi ser frem til bidrag fra lærere og andre interesserede i form af artikler, kommentarer, kritiske bemærkninger, programmer computergrafiske illustrationer osv.

Bladet er planlagt til at udkomme i hvert fald 4 gange pr. år. Næste udgave af bladet er berammet til april 1983.
Personer, der er interesserede i at deltage i det redaktionelle arbejde - skrive artikler bedes kontakte redaktionen.

Red.

Indholdsfortegnelse:

Redaktion:

Ole Karmark
Århus Tekniske Skole
Halmstadgade 6
8200 Århus N

Helge Jensen
Viborg Tekniske Skole
H.C. Andersensvej 7-9
8800 Viborg

Program til plotning af Sierpinski kurver (forside), spiraltrekant (bagside) og programmel listet i bladet tilsendes interesserede, mod fremsendelse af formatteret diskette, til redaktionen.

Kære kollega.

Vi byder hermed det nye meddelelsesblad for faget datalære og anvendelse af edb velkommen. Det er et særdeles godt initiativ gruppen her har taget.

Datalære og anvendelse af edb i undervisningen er et område, der er i rivende udvikling. For den enkelte lærer er det en vældig udfordring, men også meget tidskrævende, at følge med i udviklingen omkring maskinel, programmel og anvendelse af edb i skolesektoren.

På de tekniske skoler er der en række initiativer igang omkring udvikling af programmel og tilkobling af forskellige former for maskiner og måleinstrumenter til mikrodatamater, således at det i undervisningen i datalære og i andre fag er muligt overfor eleverne at demonstrere anvendelse af edb.

Har man prøvet at udvikle programmel eller forskellige former for interface for tilkobling af udstyr, vil man sikkert have erfaret, at det er et særdeles tidskrævende arbejde, der næsten altid tager meget længere tid end planlagt. Årsagen hertil er ofte, at man gang på gang løber ind i problemet "hvordan gøres det". Det er aldrig til at finde løsningen i manualen, det er nemlig ikke omtalt. Er man heldig, er det måske en kollega på en anden skole, der har løst problemet; ellers må man eksperimentere sig frem til en løsning.

På de tekniske skoler er der i de senere år anskaffet en hel del edb-udstyr, der er rimelig standardiseret (fortrinsvis et fabrikat). Det giver os gode muligheder for at udvikle fælles programmel til undervisningsformål. Vi har i Comal-80 (og delvis også i Pascal) et godt standard-programmeringssprog, der er velegnet til dette formål.

Der er altså ingen grund til, at hver enkelt lærer "opfinder den dybe tallerken" igen. Der er imidlertid manglet en måde at kommunikere skolerne imellem. Det nye meddelelsesblad kunne blive et af midlerne, et forum for udveksling af erfaringer omkring anvendelse af edb på de tekniske skoler.

Der er en række oplagte opgaver for "bladet": oplysninger om undervisningsforløb, anmeldelse af bøger til datalære, erfaringer med programmel, information om skoleudviklet programmel eller maskinel, o.s.v.

Fagets udvikling er afhængig af muligheden for formidling af information imellem fagets lærere. Bladets fremtid vil være afhængig af, at vi alle som lærere i faget datalære føler os forpligtet til at bidrage til denne udveksling af information.

Bjørn Andersen / Asger Svane.

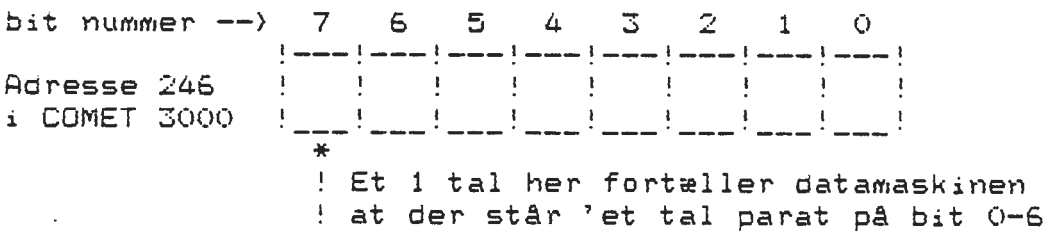
PORTE I
MIKRODATAMATER

Ordet "port", nævnes ofte i forbindelse med beskrivelsen af mikrodatamater, i relation til - printer - tastatur - dataskærm - og andre enheder med forbindelse til den ydre verden. En "port" er et sted, hvor datamaskinen kan anbringe data til og fra sine omgivelser. "Et sted" er en specifik adresse, hvor porten kan findes af datamaskinen, adressen udtrykkes som et tal. Det kan sammenlignes med en række huse, hvor hvert hus har sin adresse i form af et nummer.

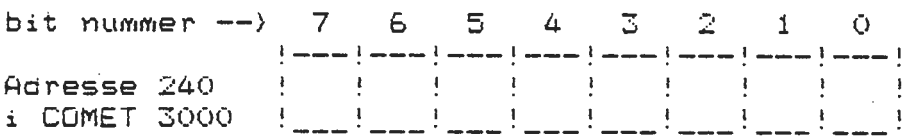
For at kunne færdes i "portenes gade" vil det være en fordel at kende indretningen af portene.

En simpel type port er en type som på forhånd er konstrueret enten som indgang eller udgang, det eneste vi behøver at vide er portens adresse og hvor bred porten er, breden på en port opgives i antal bit. Et bit er 'en plads hvor der kan indsættes 0 eller 1. Normalt har standard porte en bredde på 8-Bit, og er derved istand til at sætte 256 forskellige mønstre, af 0/1 på porten.

Et eksempel på en indgangsport er tastaturet. Her vil en aktivering af en tast afgive 'en talværdi for den pågældende tast og sætter tallet på portens indgang tallet bruger de syv af otte bit og det sidste bit bruges til at fortælle datamaskinen at der er et tal parat, hvorefter datamaskinen kan hente tallet.



Et andet eksempel kan være udgangsporten til printeren her er det datamaskinen der stiller data (tal) på porten og fortæller printeren at der er data og printeren kan så tage data fra ydersiden af porten.



Der findes mere universelle porte, hvor man på forhånd, når porten leveres, ikke har sat sig fast på om den skal fungere som ind - eller udgang eller begge dele. Den type port skal programeres af brugeren til den funktion som ønskes i den givne situation.

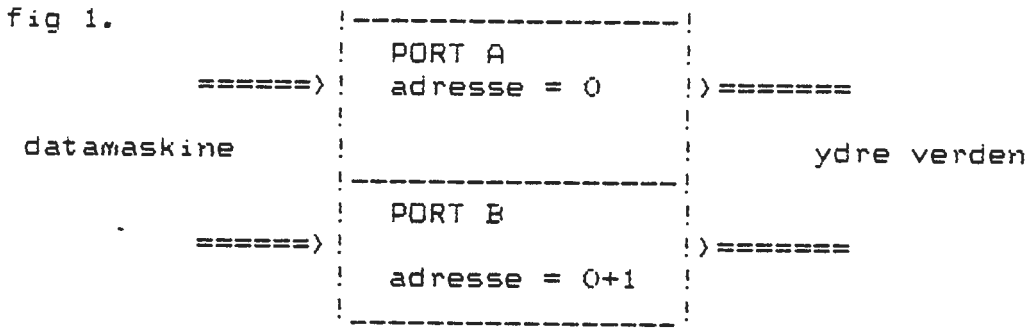
PORTE I
MIKRODATAMATER

For at kunne klare det, må porten ha' en kontrolfunktion (en portner) som fortæller de indre dele af porten hvilken aktuel funktion den skal udføre, det bliver et valg som brugeren skal tage. Kontrolfunktion skal derfor være tilgængelig, og udføres af et register med en selvstændig adresse i systemet. Dette register skal tildeles en kode som svarer til en specifik funktion.

Vi kan tage en port type Z80-PIO, og se på hvordan denne universalport kan kobles som indgang eller udgang. Der må forudsættes, at porten sidder i et system, hvor den har en grundadresse. Komponenten er udviklet så der i virkeligheden er to porte i samme enhed, nemlig port A og B.

Porten er opbygget så port-B er nabo til port-A, så hvis portens grundadresse ligger på 0, vil port-A findes på adresse 0 og port-B på adresse 1.

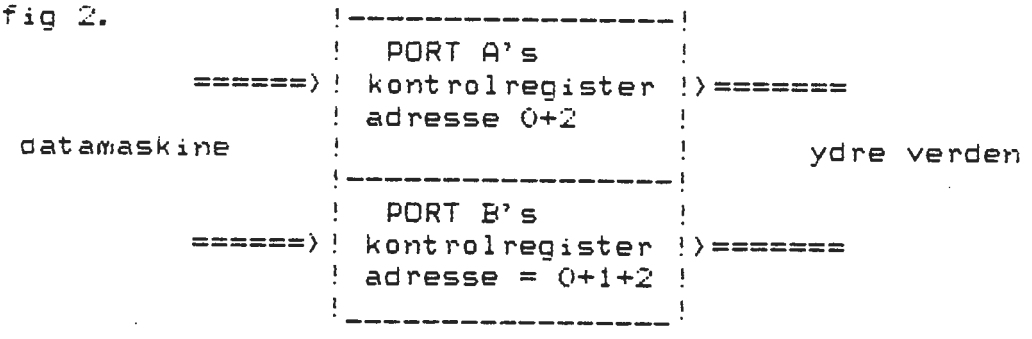
fig 1.



Kontrolregistret som betjener A og B er to selvstændige registre (en portner til hver opgang) og skal som følge heraf ha' hver sin adresse.

Fig. 2 viser at port-A's kontrolregister vil ligge på grundadressen 2 og port-B's kontrolregister på grundadressen 2+1.

fig 2.



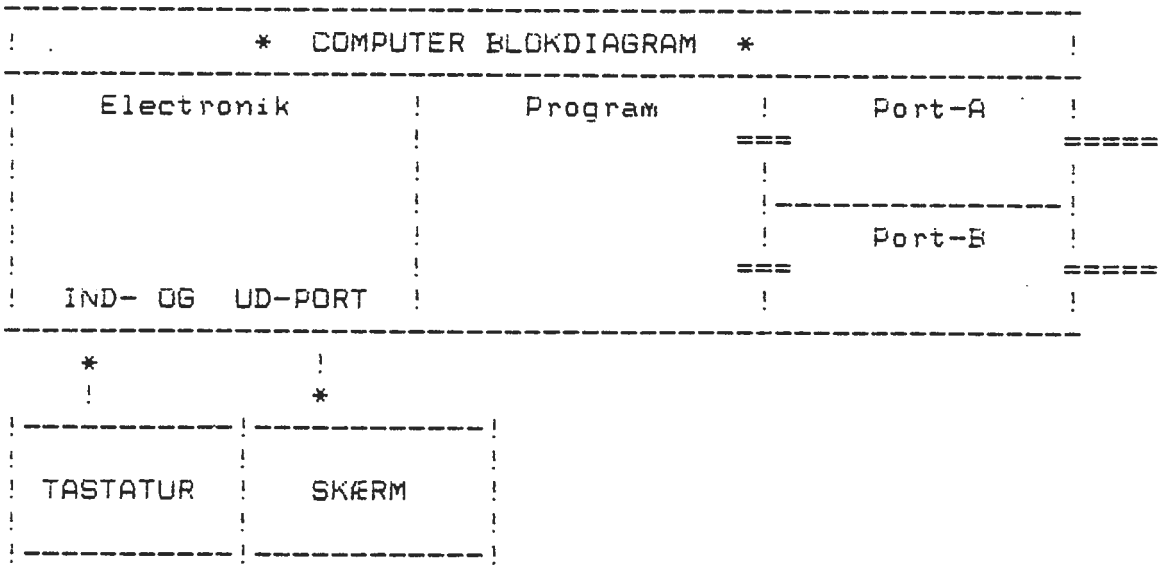
Hvis vi nu samler alle registrene i en oversigt vil det se ud som følger:

port-bredde	7	6	5	4	3	2	1	0
Grund-adresse + 0	IND ELLER UDREGISTER PORT A							
Grund-adresse + 1	IND ELLER UDREGISTER PORT B							
Grund-adresse + 2	KONTROLREGISTER PORT A							
Grund-adresse + 3	KONTROLREGISTER PORT B							

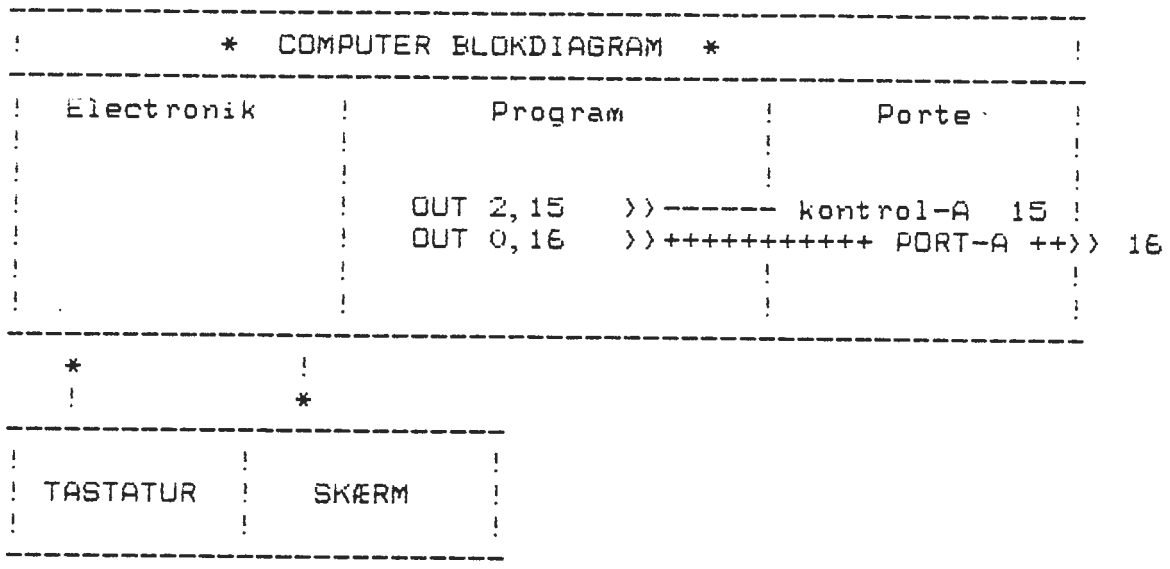
Lad os se på hvilken informationer kontrolregistret skal tilføres for at portkomponenten fungerer efter brugerens ønske.

Hvis vi vælger port-A som en udgangsport med fuld portbredde skal værdien 15 decimal tilføres kontrolregister A, hvis derimod port-A ønskes som indgang med fuld portbredde skal kontrolregister A tilføres værdien 79 decimal, det er nu talværdien som kontrolregistret tilføres der bestemmer portens funktion og ikke fabrikanten af portkomponenten.

Hvis vi stopper op her og ser på hvordan porten skal ses i forhold til datamaskinen og en bestemt opgave, kan det bedst skitseres ved en tegning og et programeksempel.



Nu er det op til brugeren hvordan portene skal fungerer, lad os betragte en situation, hvor A-porten skal fungere som udgang og tallet 16 skal overføres til den ydre verden.
Først skal tallet 15 decimal anbringes i kontrolregistret (funktion udgang) og derefter skal 16 bringes igennem porten.
Hvis sproget som anvendes er COMAL-80 findes der to instruktioner som betjener portfunktionerne, henholdsvis out- og input forkortes OUT - INP.
OUT-put opnås ved at bruge sætningen: "OUT adresse,data".
INP-ut opnås ved at bruge sætningen: "data:=INP (adresse)".
Blokdiagram med indtegnet OUT vej som viser funktionen.

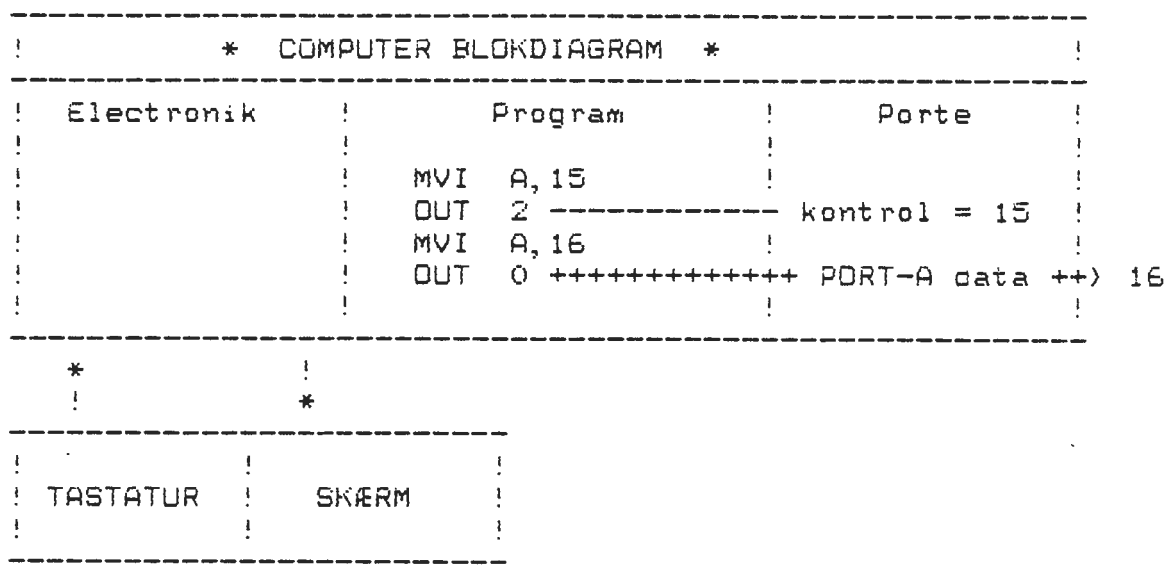


Programmet for opgaven bliver i COMAL-80 :

```
0010 DATAREGISTER:=0        // GRUNDADRESSE
0020 KONTROLREGISTER:=0+2 // GRUNDADRESSE + 2
0030 FUNKTION:=15           // UDGANGSPORT
0040 DATA:=16               // AKTUELLE DATA
0042
//*****************************************
0045 // Programmet initier PORT-A som udgang
og
0046 // bringer de aktuelle data ud (16)
0047
//-----------------------------------------
0049 //
0050 OUT KONTROLREGISTER,FUNKTION
0060 OUT DATAREGISTER,DATA
```

PORTE I
MIKRODATAMATER

Blokdiagram med indtegnet OUT vej som viser funktionen.

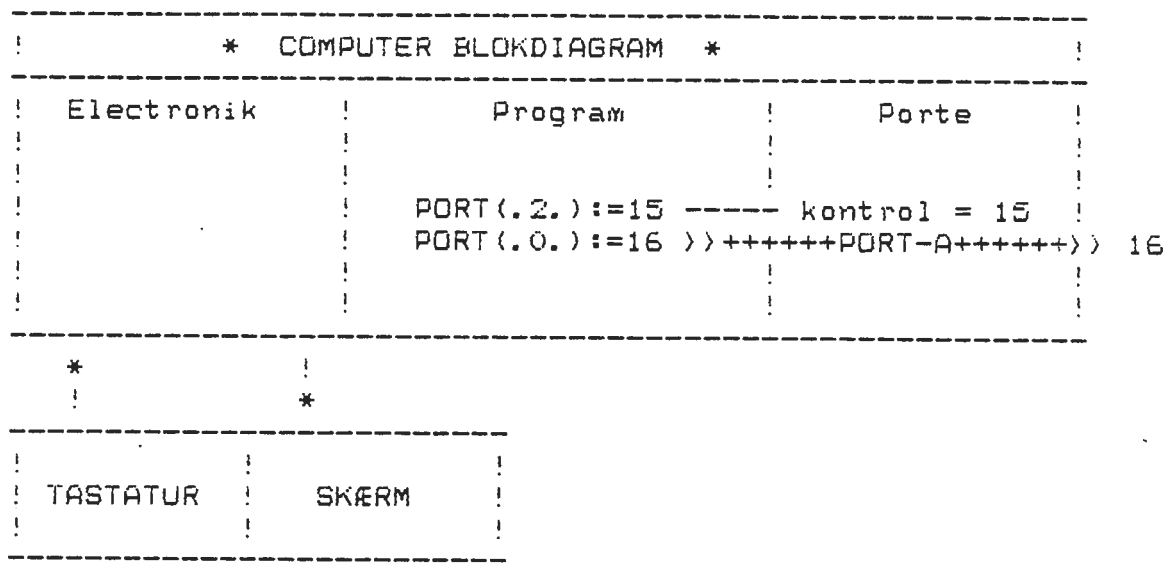


Samme løsning i assembler :

```
start:  ORG 0100H
;
datareg EQU 0000H
kontreg EQU 0000H + 2
data    EQU 16
funk    EQU 15
;
START:  MVI A,funk      ;Initier port-A
        OUT kontreg     ;som udgang
;
        MVI A,data      ;Bringer data til
        OUT datareg     ;port-A
;
        END
```

PORTE I
MIKRODATAMATER

Blokdiagram med indtegnet OUT vej som viser funktionen.



Samme løsning i COMPAS PASCAL :

```
PROGRAM port_eksempel;
const
kontrolregister = 2;
dataregister    = 0;
funktion        = 15;
data            = 16;

BEGIN
  PORT(.kontrolregister.):=funktion;
  PORT(.dataregister.):=data;
END.
```

Tallet 16 er en decimalværdi, men porten opfatter tallet binært, hvilket vil sige at portbredden er opdelt som det binære talsystem.

Tallene for portenes BIT nr.:

BIT nr. =	7	6	5	4	3	2	1	0
BIT værdi i decimal =	128	64	32	16	8	4	2	1

Det er BIT-værdien vi skal placere ude på porten for at aktivere det tilhørende BIT nr.

EKS:

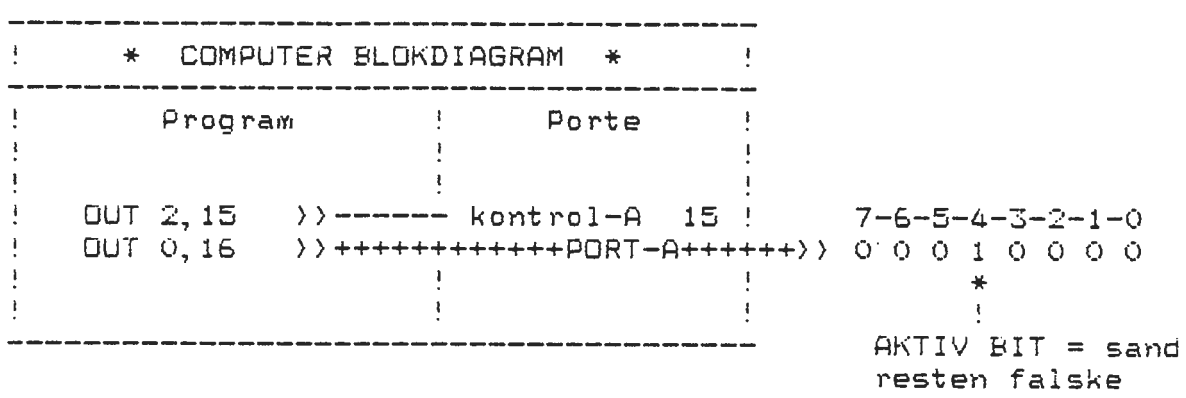
```
0010 OUT 0,128     // PORT-A BIT NR. 7 AKTIVERES = 1 //

0020 OUT 0,32      // PORT-A BIT NR. 5 AKTIVERES = 1 //

0030 OUT 0,1       // PORT-A BIT NR. 0 AKTIVERES = 1 //

0040 OUT 0,33      // PORT-A BIT NR. 5 OG 0 AKTIVERES =
//

0050 OUT 0,1+16+128 // PORT-A bit
0051 //                NR. 0,4 og 7 AKTIVERES =1


     OSV. OSV
```

Det er på den måde muligt at bruge en del af portens bredde, hvilket indebærer at hver port kan ha' otte forskellige opgaver, en for hver bit.

Hvis man ønsker at styre en el-motor, kan det lade sig gøre ved at anvende 'et bit til den pågældende funktion, og har så yderligere 7 bit til andre opgaver.

PORTE I
MIKRODATAMATER

EKS:



Hvordan vil bitmønster se ud hvis decimaltallet er :

7-6-5-4-3-2-1-0

32 = ? _ _ _ _ _ _ _ _

EKS på løsning 22 = 0 0 0 1 0 1 1 0 = 16+4+2 = 22

35 = ? _ _ _ _ _ _ _ _

156= ? _ _ _ _ _ _ _ _

255= ? _ _ _ _ _ _ _ _

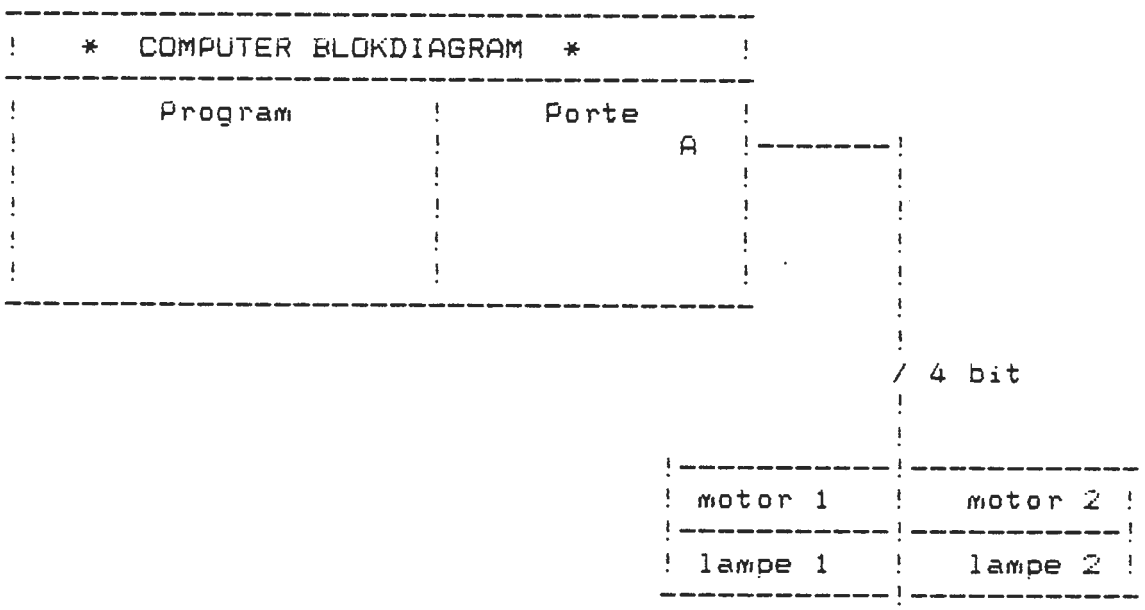
256= ? _ _ _ _ _ _ _ _

PORTE I
MIKRODATAMATER

Der findes et printkort type MPS-10, som kan placeres i COMET MPS 3000, dette kort har to Z80-PIO monteret (32 BIT I/O). På printkortet kan "grund-adressen" sættes ved hjælp af nogle bøjler (se vedlagte tegning), det bliver herved muligt at brugeren vælger en adresse som passer bedst til den stillede opgave.

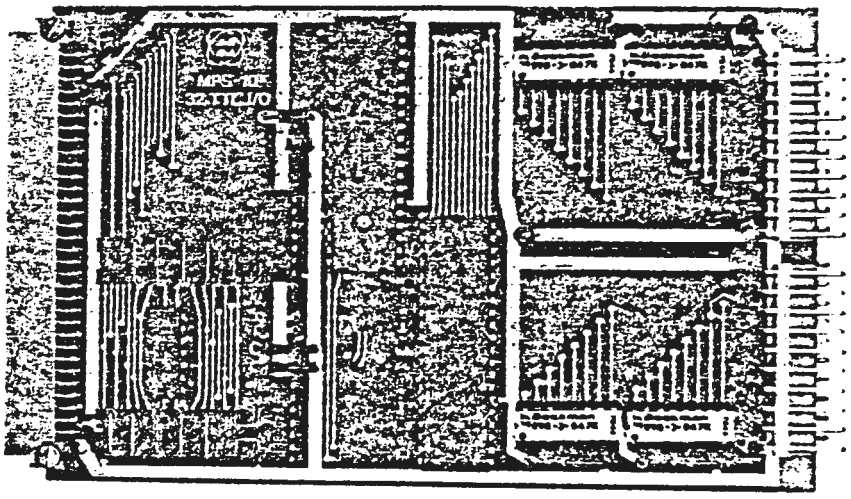
OPGAVE:

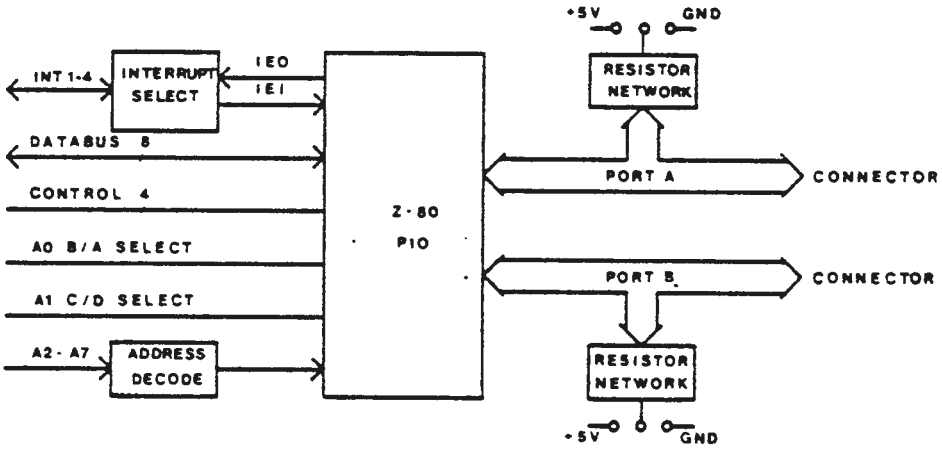
Et firma ønsker et produkt, som kan starte to motorer hvor samtidig et sæt kontrollamper lyser. Til fremstilling af produktet ønsker man at anvende et MPS-10 printkort, hvor man fastsætter bit nr. 0 og 1 til at styrer de to motorer når et bit er 0 er motoren stoppet. Kontrollamperne tildeles bit nr. 5 og 6. og lampen lyser når et bit er 1.
Fremstil et program (sprog efter eget valg) som først starter motor 1 og tænder kontrollampe, for dernæst at starte motor to og tænder kontrollampe.



Løsninger kommer næste gang.

MPS-10
32 BIT 1/0 MODUL





MPS-10 BLOCK DIAGRAM (1 OF 2 IDENTICAL BLOCKS)

MPS-10 er et 32 bit TTL-kompatibelt Input/Output modul. Det er bygget op omkring 2 stk. Z-80 PIO. Parallellinierne fra disse er ført direkte ud til 4 connectorer på modulets bagkant. De to PIO kan adresseres u-afhængigt af hinanden.

Dansk kvalitet

HH-Electronic ApS.
Højvangen 6
3480 Fredensborg
Tlf. (03) 28 38 41

Adressering.
Adresseringen af hver af modulets PIO-blokke foretages ved strapning af adresselinierne A2-A7, hvilket tillader anvendelse af max. 64 PIO-blokke (eller andre MPS-moduler) i et system. A0 benyttes til valg af A- eller B-port i PIO. A1 benyttes til valg af Control- eller Data-mode (se PIO manual).
Strapning af blokadressen sker i soklerne i modulets nederste venstre hjørne (fig. 1). Skal blokken reagere på "1" på en adresselinie, skal der udføres en HH strapning. Skal blokken reagere på "0" på en adresselinie, skal der udføres en LL strapning.
Eksempel: blokken ønskes placeret på adressen (A2-A7): 100100. Strapning foretages som vist på fig. 2.

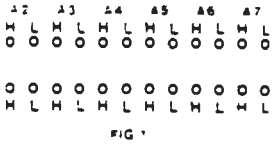


FIG. 1

Interrupt.
Kobling af blokkens placering i "daisy-chain" systemet sker ved strapning af 8-bens sokkel i modulets venstre side, som vist på fig. 3. For blokken med den højeste interruptprioritet strappes IEI til +5 V (strapning på print nedenfor sokkel). I øvrigt henvises til PIO manual.

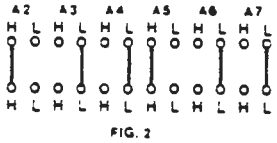


FIG. 2

Power : 5 V fra MPS-bus, max. 200 mA.
Format: Europakort (100 x 160 mm) med 64-polet DIN 41612 connector til MPS-bus og Molex connectorer på modulets bagkant.

Tilslutninger.
På modulets bagkant findes 4 connectorer (2 på hver side), idet de to A-portes connectorer sidder på komponentsiden og B-portenes connectorer sidder på undersiden. Connectorforbindelserne fremgår af fig. 4.
I forbindelse med hver port er der et modstandsnetværk (normalt 8 stk. 100 kohm) som vist på fig. 5. Modstandenes fællespunkt kan v.h.a. en strapning forbindes til GND eller +5 V efter ønske (se fig. 6).

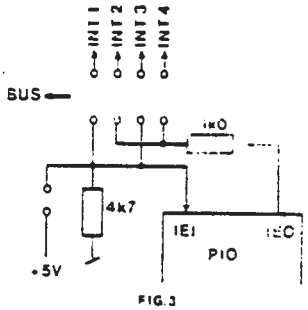


FIG. 3

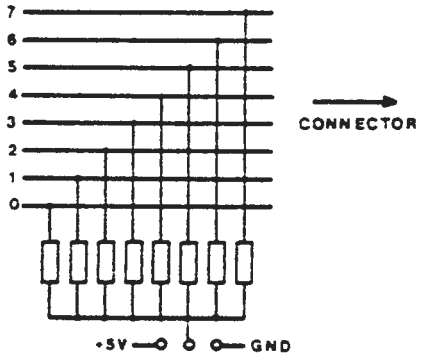


FIG. 5

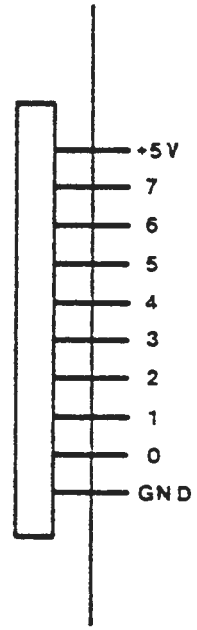


FIG. 4

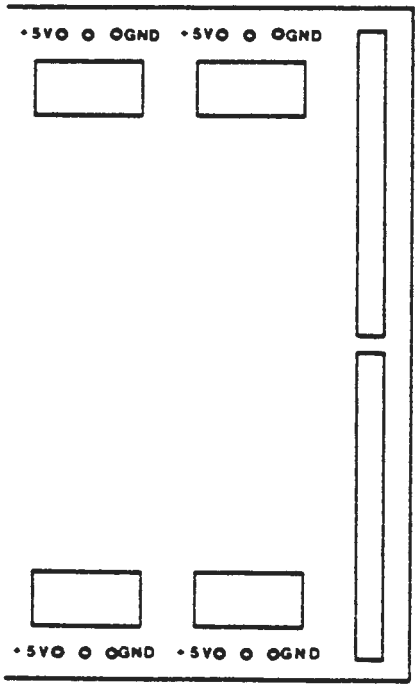


FIG. 6

KOMBINATORISKE ALGORITMER I

Ved løsning af mange praktiske problemstillinger har man ofte brug for algoritmer, der er beregnet på at udføre kombinatoriske processer, dvs. fremstille forskellige kombinationer af en på forhånd bestemt endelig mængde af elementer. Sådanne processer er det ofte kun muligt at fremstille ved anvendelse af computere, idet antallet af mulige kombinationer - endog med et mindre antal elementer - hurtigt når eksplosive højder.

Det gælder for de indledende problemstillinger nemlig:1)generering af samtlige delmængder af en mængde og 2) generering af samtlige permutationer af en mængde, at de matematisk set er ret enkle at forstå og begribe, hvorimod det ved kravet om en fuldstændig algoritmisk beskrivelse af problemet udmærket kan volde vanskeligheder. Arbejdet med at beskrive og derved begribe processen er dog lønnende, idet det må fremhæves, at løsningsmetoderne illustrerer generelt anvendelige teknikker og programmeringsmetoder.
Yderligere bemærkes det, at opgaver af kombinatorisk natur, dvs. manipulationer på tal og talsæt, er særdeles velegnede til løsning ved hjælp af en datamat.

Generelt om kombinatoriske problemstillinger kan man opstille følgende spørgsmål:

1. Hvor mange forskellige muligheder eksisterer for et givet antal elementer?

2. Hvordan ser de ud?, dvs. kravet om generering, listning, udskrivning eller opbevaring af samtlige muligheder.

3. Evaluering af de enkle muligheder, herved skal forstås en vurdering af de enkle kombinationer om de tilfredsstiller bestemte krav. Følgende eksempel kan belyse problemstillingen: En handelsrejende skal besøge ti forskellige byer. Det må formodes, at han vil være interesseret i at få oplyst i hvilken rækkefølge, det er mest fornuftig at besøge byerne, for at minimere rejsetiden. Bemærk at antallet af mulige rejseruter er 10!=3.628.800. Indlæses afstandene mellem de enkelte byer i datamaten, kan programmet bagefter evaluere på hver enkelt rejserute ved generering af alle rækkfølger og således afgøre hvilken rute, der er den hurtigste (billigste). Et andet eksempel kunne være at generer samtlige tipsrækker (3^13 rækker eller 1.594.320) og evaluerer på disse ud fra bestemte betingelser, opstillet på baggrund af forventninger om de mest sandsynlige rækker for ugens tipskupon.

I det følgende er det navnlig spørgsmålene om hvor mange og hvordan ser de ud, der vil blive behandlet. Spørgsmålet om evaluering af de enkelte kombinationer hører ind under de

særlige anvendelsesområder og er dermed knyttet til den konkrete anvendelsessituation. Af disse årsager overlades dette til den enkelte læser.

DELMÆNGDER AF EN MÆNGDE.

Antallet af delmængder.

Den første kombinatoriske problemstilling, der skal behandles, er spørgsmålet om hvormange delmængder, der kan dannes af en endelig mængde af elementer.

Vi kan definere mængden M således:

M = (A, B, C, D)

hvor mængden M består af fire forskellige elementer. Når vi skal bestemme antallet af delmængder, der kan dannes af M, kan vi i hvert fald sige at det gælder for et element i M - f.eks. elementet A - at enten er det med i delmængden eller osse er det ikke med, dvs. der eksisterer to muligheder for hvert element.
Skitseres opgaven som gående ud på at udfylde følgende rubrikker:

Elementer:

A	B	C	D

ses det, at for hvert element er der netop 2 muligheder: enten er det med eller ikke med. Dette medfører, at antallet af mulige delmængder af mængden M er: 2*2*2*2 = 16 eller 2^4. Det skal bemærkes, at vi her osse har medregnet den tomme mængde.

Er antallet af elementer i M eksempelvis 8 får vi antallet af mulige delmængder er 2^8 = 256.

Generelt gælder:
Regel 1

Antallet af delmængder af en N-mængde er 2^N, hvor N er antallet af elementer i mængden.

Vi har nu besvaret det første spørgsmål om hvormange muligheder, der eksisterer for et givet antal elementer. Det andet spørgsmål skal besvares i det følgende.

KOMBINATORISKE ALGORITMER

Algoritme til generering af alle delmængder.

Vi vender os nu til spørgsmålet om, hvorledes det er muligt at udskrive eller generere samtlige delmængder af en mængde.

Opgaven kan beskrives således:

-konstruer et program, der indlæser antallet af elementer i en mængde. På baggrund af denne mængde angivelse skal programmet fremstille (udskrive) samtlige delmængder af denne mængde.

Løsningen på opgaven vil blive beskrevet på følgende måde: 1) først angives ved et eksempel en metode, der kan anvendes. 2) Dernæst udvikles på baggrund af eksemplet en algoritme, der løser opgaven. 3) Endelig angives et program formuleret i COMAL-80, et program, der udskriver samtlige delmængder. Lad os vende tilbage til eksemplet, hvor M = (A, B, C, D). Beregningen af antallet af delmængder skete fuldstændig uafhængigt af hvilken type elementer mængden M bestod af. Overvej følgende opgave: Hvormange tal i totalsystemet kan man skrive, hvis man højst må anvende 4 cifre ?

Svaret er 2^4 = 16, hvor det mindste tal er 0000(2) og det største er 1111(2) (her angivet i totalsystemet). Antallet kan beregnes på samme måde, som ved beregning af antallet af delmængder af en mængde (Regel 1). Ved opstilling af en algoritme til udskrivning af samtlige delmængder kan vi udnytte samme fremgangsmåde som ved beregning af antallet. F.eks. vil tallene

1001 svarer til delmængden (A D)
1011 svarer til delmængden (A C D)
osv.

Til frembringelse af disse tal (bitmønstre) benytter vi en vektor (et talsæt) og antager at samtlige af vektorens elementer ved start er nul.

Bitnr.	1	2	3	4	
Vektor.	0	0	0	0	(Svarer til den tomme mængde)

Når vi har gennemløbet samtlige muligheder vil vi have følgende billede:

Bitnr.	1	2	3	4	
Vektor.	1	1	1	1	Svarer til delmængden (A, B, C, D) hvor der er 4 ettaller

og vi er færdige.

Problemet er nu systematisk, at generere de mellemliggende bitmønstre.

1. Billed (start)

Bitnr.	1	2	3	4	
Vektor.	0	0	0	0	( )

2. Billed

Bitnr.	1	2	3	4	
Vektor.	1	0	0	0	(A)

3. Billed

Bitnr.	1	2	3	4	
Vektor.	0	1	0	0	(B)

(Mente overførsel)

4. Billed

Bitnr.	1	2	3	4	
Vektor.	1	1	0	0	(A, B)

5. Billed:

Bitnr.	1	2	3	4	
Vektor.	0	0	1	0	(C)

(Mente overførsel)

osv.

På grundlag af ovenstående eksempel (billedforløb) kan processen beskrives ved følgende algoritme. I beskrivelsen af algoritmen antager vi, at vi har proceduren UDSKRIV tilrådighed. Et eksempel på en sådan procedure kan ses i programudskriften på en efterfølgende side. Om algoritmens variable kan det nævnes, at bitmønstrene opbevares i talsættet VEKTOR. Variablen BITNR optræder som indeks for elementerne i talsættet. Variablen ANTALETTALLER opbevarer hele tiden det aktuelle antal ettaller, der findes i talsættet.
Læseren opfordres til at afprøve algoritmen med et konkret eksempel, eventuelt kan man kontrollerer sin forståelse af algoritmen ved at gennemgå foranstående eksempel.

Algoritme delmængde

```
indlæs   antal ( antal elementer i mængden)
ooret    talsættet vektor
sæt      vektor elementerne til 0
sæt      antalettaller til 0
udskriv  delmængde
sæt      slut til falsk
sålænge  slut forskellig fra sand
udfør    sæt bitnr lig 1
         næstebit:
              hvis   vektor(bitnr) = 0
              så     sæt vektor(bitnr) til 1
                     øc antalettaller med 1
                     hvis antalettaller lig antal
                     så   sæt slut til sand
              ellers (*menteoverførsel*)
                     sæt vektor(bitnr) til 0
                     mindsk antalettaller med 1
                     øg bitnr med 1
                     gå til næstebit
              udskriv  delmængde
```

Ovenstående algoritme kan umiddelbart omformuleres til et COMAL-80 program. Af denne årsag er det følgende program DELMÆNG ikke gengivet, som den viste algoritme, men programmet fremtræder med en noget anderledes struktur. Denne struktur giver programmet flere anvendelses muligheder. I programmet genfindes den væsentligste del af algoritmen delmængde i proceduren ALLEDELE.

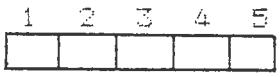
```
0005  // program delmæng
0010 PROC ALLEDELE(NM)
0020  // INITIALISER OG NULSTIL VEKTOR.
0030  // DEN TOMME MÆNGDE KAN DEREFTER UDSKRIVES.
0040  //
0050  IF FØRSTE THEN
0060   FOR BITNR:=1 TO NM DO
0070    VEKTOR(BITNR):=0
0080   NEXT BITNR
0090   ETTALLER:=0
0100   FLERE:=TRUE
0110   FØRSTE:=FALSE
0120   GOTO UD
0130  ENDIF
0140  //
0150  //
0160  BITNR:=1
0170 LABEL NÆSTEBIT
0180  IF VEKTOR(BITNR)=0 THEN
0190   VEKTOR(BITNR):=1
0200   ETTALLER:=ETTALLER+1
0210   // DET UNDERSØGES OM DET ER DEN SIDSTE DELMÆNGDE
0220   IF ETTALLER=NM THEN FLERE:=FALSE
0230  ELSE
0240   // MENTE OVERFØRSEL
0250   VEKTOR(BITNR):=0
0260   ETTALLER:=ETTALLER-1
0270   BITNR:=BITNR+1
0280   GOTO NÆSTEBIT
0290  ENDIF
0300  //
0310 LABEL UD
0320  //
0330 ENDPROC ALLEDELE
0340 //
0350 PROC UDSKRIV
0360  PRINT "(";
0370  FOR K:=1 TO N DO
0380   IF VEKTOR(K)=1 THEN PRINT ALFA$(K);
0390  NEXT K
0400  PRINT ")"
0410 ENDPROC UDSKRIV
0420 // INITIALISERING AF ALFABET
0430 DIM ALFA$ OF 28
0440 FOR K:=1 TO 28 DO
0450  READ ALFA$(K)
0460 NEXT K
0470 DATA "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N"
0480 DATA "O", "P", "Q", "R", "S", "T", "U", "V", "X", "Y", "Z", "Æ", "Ø", "Å"
0490 //
0500 //
0510 // HOVEDPROGRAM
0520 //
0530 PRINT "UDSKRIVNING AF SAMTLIGE DELMÆNGDER AF EN MÆNGDE."
0550 INPUT "ANGIV ANTALLET AF ELEMENTER I MÆNGDEN: ": N
0560 PRINT "ANTALLET AF DELMÆNGDER AF ";N;"-MÆNGDEN ER ";2^N
0570 DIM VEKTOR(N)
0580 //
0590 FØRSTE:=TRUE
0600 REPEAT
0610  EXEC ALLEDELE(N)
0620  EXEC UDSKRIV
0630 UNTIL NOT FLERE
0640 //
0650 END
```

PERMUTATIONER AF EN MÆNGDE

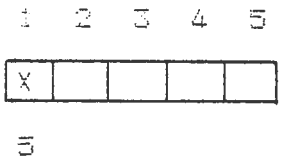
Antallet af permutationer af en mængde

Vi har givet en mængde af forskellige elementer f.eks. M = (? X $ * #), hvor mange forskellige rækkefølger eller permutationer kan vi danne af mængden M, når det gælder at hvert element kun må forekomme en gang og alle elementer skal anvendes i hver permutation?

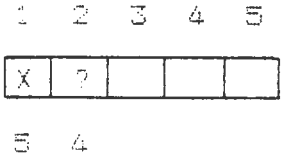
Svaret er 5!, som betyder 5*4*3*2*1 og læses 5 fakultet eller 5 udråbstegn. At det forholder sig således indses let af følgende eksempel:



Opgaven går nu ud på at vælge et element til hver rubrik. Ved placeringen af et element i den første rubrik har vi 5 elementer at vælge imellem, altså 5 muligheder. Vi vælger elementet 'X'.



Ved placering af et element i næste rubrik har vi 4 elementer at vælge imellem (vi har brugt et!), der er altså 4 muligheder. Vi vælger elementet '?'.



Ved at gentage udvælgelsesprocessen, når vi til at antallet af måder vi kan udvælge rækkefølger eller permutationer på er 5! = 5*4*3*2*1 = 120.

Generelt gælder følgende regel:

Regel 2

> Antallet af permutationer af en mængde M bestående af N forskellige elementer er:
>
> N! = N * (N-1) * (N - 2) * .... * 3 * 2 * 1

Permutations algoritme

Ved permutering af en mængde, er det første spørgsmål, der dukker op, hvordan kan man skabe en eller anden form for ordning af de forskellige permutationer?. En sådan ordning er nødvendig, hvis metoden skal fremstille permutationerne systematisk. Ved permutering af mængder M = (1,2,3,4,5) kan man istedet for at betragte elementerne som et talsæt, opfatte dem som et tal, nemlig den numeriske størelse: 12345. Man kan notere sig at den rækkefølge cifrene fremstår i, samtidig er det mindste tal det er muligt at skrive med disse cifre. I det følgende skal vi udvikle en metode der systematisk genererer permutationer af mængden M. Metoden bygger på følgende principper: Vi begynder med den første permutation, hvor talsættets elementer står i en stigende orden (1,2,3,4,5). Når denne permution er genereret udskrives den. Den næste permutation udvikles nu på baggrund af den foregående (den første), idet vi nu opfatter talsættet som et tal og ombytter cifrene så det næste tal i den stigende orden, vi opbygger, dannes. Vi skal altså fremstille det næstmindste tal.

Det følgende eksempel viser en del permutationer for M = (1,2,3,4,5).

Permutationer:	Ombytning af cifre:
1.: 1 2 3 4 5	1 2 3 4 5 → 1 2 3 5 4
2.: 1 2 3 5 4	1 2 3 5 4 → 1 2 4 5 3 → 1 2 4 3 5
3.: 1 2 4 3 5	1 2 4 3 5 → 1 2 4 5 3
4.: 1 2 4 5 3	1 2 4 5 3 → 1 2 5 4 3 → 1 2 5 3 4
5.: 1 2 5 3 4	1 2 5 3 4 → 1 2 5 4 3
6.: 1 2 5 4 3	1 2 5 4 3 → 1 3 5 4 2 → 1 3 2 4 5
7.: 1 3 2 4 5	1 3 2 4 5 → 1 3 2 5 4
8.: 1 3 2 5 4	1 3 2 5 4 → 1 3 4 5 2 → 1 3 4 2 5
9.: 1 3 4 2 5	1 3 4 2 5 → 1 3 4 5 2
-	
-	
-	
120.: 5 4 3 2 1	

Hvornår er vi færdig med processen? En måde at afgøre problemet på er, at beregne hvormange permutationer, der findes af mængden. En enklere metode er at sige, at det er, når vi har udskrevet det største tal, man kan danne af

mængden, dvs. (5 4 3 2 1). Vi kan heraf slutte, at når elementerne står i omvendt orden, er vi færdige (modsat startorden). Dette forhold kan vi udnytte ved beskrivelse af processen, idet vi blot stopper, når cifrene står i omvendt orden.

algoritmen

Som ved delmængde ~~afgørelsen~~ formulerer vi problemet.

- konstruer et program, der indlæser antallet af elementer i en mængde. Programmet genererer dernæst samtlige permutationer af mængden.

Med udgangspunkt i ovennævnte eksempel vil vi nu udvikle og beskrive en algoritme til generering af permutationer.

Indlæs	antal (elementer der skal permuteres)
Opret	talsættet vektor
Sæt	vektor lig 1, 2, 3, 4, 5
Sæt	flere lig sand
Sålænge	flere lig sand
Udfør	udskriv permutation generer næste permutation

Hermed har vi vist hovedstruktufren i algoritmen. Det største problem står dog tilbage, nemlig delalgoritmer: generer næste permutation.

Vi tager udgangspunkt i følgende permutation, og opgaven er nu at udvikle næste permutation:

	1	2	3	4	5
Vektor	1	2	5	4	3

Vi kan umiddelbart se (eksemplet), at vi skal ombytte vektor(5) og vektor (2). Fremgangsmåden er at vi begynder med elementet vektor(5) - med indholdet 3 - og undersøger om dette er større end det foranstående (vektor(4)) - dette er ikke tilfældet, og vi undersøger så om vektor(4) er større end det foranstående - vektor(3) - heller ikke, og vi undersøger så om vektor(3) er større end vektor(2) - ja dette er tilfældet. Vi ved nu at vektor(2) med indholdet 2 skal ombyttes, men med hvad? Vi gemmer indholdet af vektor(2) i hjælpe variablen ombyt. - Vi begynder igen at gennemløbe vektoren bagfra, men undersøger nu om vektor(5) er større end indholdet af ombyt - ja dette er tilfældet. Det er nu disse elementer, der skal ombyttes, altså vektor(2) og vektor(5). Vi får da følgende delalgoritme:

```
for i:=antal nedtil 2
udfør  hvis vektor(i) > vektor(i-1)
       så  ombyt:=vektor(i-1)
           for j:=antal nedtil i
               udfør  hvis vektor(j) > ombyt
                      så  vektor(i-1):=vektor(i-1)
                          vektor(j):=ombyt
                          (* del den del af elementerne *)
                          (* der står til højre for vek-*)
                          (* toren i to og ombyt ele-   *)
                          (* menterne parvis!           *)
```

Efter udførelse af denne del af algoritmen har vi

	1	2	3	4	5
vektor	1	3	5	4	2

Som vi kan se ud fra eksemplet, er vi endnu ikke helt færdige. Vi mangler at vende den del, der står til højre for vektor(2) om. Det overlades til læseren som øvelse, at beskrive denne del af processen.

Problemet om, hvornår vi er færdige, løses som det ses, på den måde at når vi har gennemløbet vektoren uden at have fundet et foranstående element, der er mindre, stopper processen.

Nedenfor angives en fuldstændig algoritme til løsning af problemet.

```
indlæs  antal (antal elementer, der skal permuteres)

opret   talsættet vektor

sæt     vektor lig 1,2,...antal

sæt     flere lig sand

sålænge [flere lig sand]

udfør   udskriv permutationer

        for i:=antal nedtil 2

        udfør  hvis [vektor(i) > vektor(i-1)]

               så   ombyt:=vektor(i-1)

                    for j:=antal nedtil i

                    udfør hvis [vektor(j) > ombyt]

                          så   (* ombytning *)

                               vektor(i-1):=vektor(j)

                               vektor(j):=ombyt

                               (* vend højre side om *)

                               for k:=0 optil (antal-i-1) div 2

                               udfør [ombyt:=vektor(antal-k)

                                      vektor(antal-k):=vektor(i+antal)

                                      vektor(i+K):=ombyt]

                               gå til ud

        sæt     flere lig falsk

        ud:
```

```
0010 //  program permutat
0015 //
0020 PROC PERM(NM)
0030  //
0040  IF FØRSTE THEN
0050   FOR I:=1 TO NM DO
0060    VEKTOR(I):=I
0070   NEXT I
0080   FØRSTE:=FALSE
0090   FLERE:=TRUE
0100   GOTO UD
0110  ENDIF
0120  //
0130  FOR I:=NM DOWNTO 2 DO
0140   IF VEKTOR(I))VEKTOR(I-1) THEN
0150    OMBYT:=VEKTOR(I-1)
0160    //
0170    FOR J:=NM DOWNTO I DO
0180     IF VEKTOR(J))OMBYT THEN
0190      //  OMBYT
0200      VEKTOR(I-1):=VEKTOR(J)
0210      VEKTOR(J):=OMBYT
0220      //
0230      //  VEND HØJRESIDE OM
0240      FOR K:=0 TO (NM-I-1) DIV 2 DO
0250       OMBYT:=VEKTOR(NM-K)
0260       VEKTOR(NM-K):=VEKTOR(I+K)
0270       VEKTOR(I+K):=OMBYT
0280      NEXT K
0290      //
0300      GOTO UD
0310      //
0320     ENDIF
0330    NEXT J
0340    //
0350   ENDIF
0360  NEXT I
0370  //
0380  FLERE:=FALSE
0390  //
0400 LABEL UD
0410  //
0420 ENDPROC PERM
0430 //
0440 PROC UDSKRIV
0450  TÆLLER:=TÆLLER+1
0460  PRINT USING " ###: ": TÆLLER,
0470  FOR P:=1 TO LÆNGDE DO
0480   PRINT TEKST$(VEKTOR(P));
0490  NEXT P
0500  IF NOT (TÆLLER MOD SØJLE) THEN PRINT
0510 ENDPROC UDSKRIV
0520 //
```